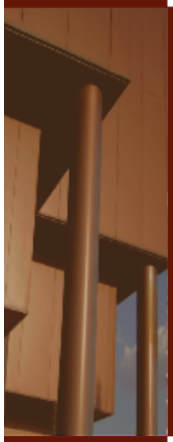
Proyecto Fin de Grado
Grado en Ingeniería de las Tecnología de Telecomunicación

Sistemas de localización basados en 802.11

Autor: José Manuel Noguerol Díaz
Tutor: Pablo Aguilera Bonet
Tutora ponente: Eva María Arias de Reyna Domínguez



Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Sistemas de localización basados en 802.11

Autor:
José Manuel Noguerol Díaz

Tutor:
Pablo Aguilera Bonet
Profesor colaborador
Tutora:
Eva María Arias de Reyna Domínguez
Profesora titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Sistemas de localización basados en 802.11

Autor: José Manuel Noguerol Díaz

Tutor: Pablo Aguilera Bonet

Tutora: Eva María Arias de Reyna Domínguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A mis padres por ser el pilar fundamental que me ha poyado y permitido seguir con mis estudios. Por todos sus esfuerzos durante estos años, por ayudarme a crecer y levantarme cuando los tiempos no eran los mejores y más lo necesitaba. A mi hermano, el otro yo, un reflejo que me ha alegrado cada instante de este recorrido. A mi tía, por hacerme ver todo de una manera diferente y positiva. A todos ellos, gracias por no haber permitido que me rindiera y, sobre todo, gracias por confiar en mí.

A mis abuelos, mi pilar emocional. Sé que siempre habíais soñado con que llegara este día. Espero que desde ahí arriba me acompañéis y me deis mucha fuerza.

A mi tutor, Pablo Aguilera Bonet, por haberme brindado la oportunidad de realizar este excelente proyecto y permitirme profundizar en esta temática tan innovadora. A Eva María Arias de Reyna Domínguez, por permitirnos que este proyecto haya llegado a su finalización y presentación.

A todos esos nuevos y maravillosos compañeros que la vida me ha dado la oportunidad de conocer durante estos años. Vuestro apoyo y nuestras aventuras han sido muy determinantes, gracias.

A mis compañeros de empresa, por hacerme concebir la vida de otra manera y enseñarme tanto. Gracias por vuestra paciencia.

A mis amigos, sin duda, una de las fuentes de apoyo y motivación más importante. En especial a Antonio Nieto y Antonio Javier Nieto, por vuestro respaldo incondicional en todo momento. Os debo mucho. De corazón, gracias.

José Manuel Noguerol Díaz

Sevilla, 2018

Resumen

En este proyecto se va a crear un sistema autónomo capaz de analizar los diferentes algoritmos de localización para el estándar 802.11 llevando a cabo un conjunto de simulaciones y aplicaciones que nos permitan una toma de decisiones y conclusiones de manera sencilla y eficaz. Con ello, seremos capaces de determinar con mayor o menor certeza, si estos métodos son adecuados para aplicarlo en escenarios reales.

Para lograr lo anterior, todas las pruebas realizadas irán acompañadas de un conjunto de gráficos o tablas de manera que sea sencillo analizar los resultados obenidos.

Con el objetivo de obtener las conclusiones finales, analizar la escalabilidad, detectar fallos y situaciones críticas se variarán los parámetros de las simulaciones y se añadirá interferencia, desvanecimiento o ruido.

Este se proyecto estará acompañado de una sencilla aplicación, que permitirá gestionar en detalle cada uno los elementos de las simulaciones (número y localización de dispositivos, tamaño del mapa, parámetros de simulaciones…), permitiendo a su vez, la continuación y mejora de este proyecto gracias a la modularidad del código realizado.

Abstract

The aim of this proyect is to create an autonomous system capable of analyzing the different existing location algorithms for the 802.11 standar, performing a set of simulations that allow us to make decisions and conclusions in a simple and effective way. So that, we will be able to determine if these methods are suitable to be applied in real scenarios.

In order to achieve that, all the tests carried out will be accompanied by a set of graphs or tables, so that it is easy to analyze the results that have been got.

To obtain the final conclusions, analyze the scalability, detect failures and critical situations, the parameters of the simulations will be varied and interference, fading or noise will be added.

Eventually, I created a simple application, which will allow to manage in detail each element of the simulations (number and location of devices, map size, simulation parameters ...), allowing the continuation and improvement of this project because of the modularity of the code made.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

"Incluso la gente que afirma que no podemos cambiar nuestro destino, mira antes de cruzar la calle."

- Stephen Hawking -

El avance tecnológico está abarcando cada vez más todos los aspectos de nuestra vida cotidiana con el fin de sobrellevarla de una manera más controlada, sencilla, segura y eficiente. La seguridad e Internet de las Cosas quizás han sido los aspectos en los que se han hecho especial hincapié estos últimos años con el objetivo de tener un mayor control de todo lo que nos rodea.

Estimar la propia posición siempre ha sido un problema en la historia humana. Si la necesidad de obtener la posición de los seres humanos y los objetos ha sido ha sido requerido por la sociedad, se hace evidente que, en el futuro, la importancia de la localización crecerá. Los avances en sensores y electrónica permiten nuevas aplicaciones que demandan información de ubicación precisa en tiempo real a bajo costo. Además, muchas de estas aplicaciones se encuentran en interiores, ya que las personas y las máquinas tienden a pasar cada vez más tiempo dentro de edificios.

Los actuales sistemas de posicionamiento global no permiten la localización con precisión de dispositivos en interiores, y requieren un equipamiento más costoso y con un hardware especifico. Debido a esto, hay una clara necesidad de desarrollar sistemas de posicionamiento o localización en interiores utilizando infraestructuras ya existentes.

1.1 Motivación

Durante mi estancia de prácticas en una empresa de Sevilla, Pablo Aguilera, exprofesor de la Universidad de Sevilla, me ofreció la posibilidad de realizar un proyecto basado en la localización WiFi. Siempre había sido un campo que me llamó mucho la atención, sobre todo los algoritmos utilizados para localizar.

Sin duda alguna, acepté realizar este proyecto e inicié mi aventura en un mundo totalmente desconocido hasta ahora, siempre dispuesto a aprender y aportar mi grano de arena.

1.2 Objetivos

El objetivo principal de este proyecto es crear sistema capaz de analizar y comparar los parámetros más relevantes para los diferentes métodos de localización existentes aplicados al estándar 802.11 [1].

Además de esto, se pretende crear una herramienta que permita la utilización de estos métodos en escenarios reales de manera que cualquier profesional pueda realizar pruebas y tomar decisiones acerca de cuál método es

más conveniente para su caso.

Como se han concluido en varias publicaciones la importancia de la localización WiFi está en crecimiento e incluso haciendo redundante algunas ya existentes o llegando a combinarse para ofrecer una mayor precisión [2].

Otro punto clave, dentro los objetivos, es mantener la seguridad de los usuarios conectados a cualquier punto de acceso sin comprometer ninguno de sus datos, así como cualquiera de sus actividades.

Y como último punto a destacar, se han realizado numerosas pruebas para que todas las conclusiones obtenidas sean aplicadas para cualquier número de puntos de accesos y usuarios para la máxima cantidad posible de escenarios, es decir, que sea lo más escalable posible.

Como resumen, los objetivos que se pretenden conseguir son:

1. Explicación y comparación de los diferentes algoritmos de localización, así como de los métodos de posicionamiento.
2. Herramienta para aplicar estos algoritmos a diferentes escenarios y obtener conclusiones fácilmente.
3. Mantener seguridad del usuario sin comprometer sus datos ni su tráfico.
4. Máxima escalabilidad para garantizar que este proyecto se pueda aplicar al mayor número de escenarios posibles, independientemente de cualquier otro aspecto.

1.3 Plan de trabajo

La realización de este proyecto se puede organizar en varias fases, las cuales comprenden aspectos de investigación, diseño, implementación, pruebas y documentación. Dichas fases se enumeran a continuación:

1. Estudio del estado del arte, donde se valoran los distintos métodos de localización
2. Concepción de la idea, estudiando los diferentes algoritmos en profundidad.
3. Implementación de los diferentes métodos de localización en Python [3].
4. Realización de simulaciones más completas para obtener conclusiones y resultados finales.

1.4 Estructura de la memoria

En este apartado se pretende comentar la organización y estructura de los contenidos de la memoria, aportando una breve descripción de cada uno de ellos.

1. **Introducción**: Primer capítulo en el que se introduce la idea, motivaciones, objetivos, plan de trabajo y estructura de la memoria.
2. **Estado del arte**: En este capítulo se entra en profundidad en los métodos utilizados en este trabajo, además de comentar las diferentes alternativas que ofrece el mercado para cubrir nuestras necesidades
3. **Diseño**: En este punto se detalla la estructura del sistema, así como la finalidad y el uso de las distintas tecnologías que se usarán en éste. También se indican los motivos que han llevado a la elección de cada elemento de este proyecto.

4. **Implementación**: Se incluyen todos los aspectos del desarrollo del sistema, acorde al diseño indicado
5. **Pruebas y simulaciones**: En este apartado se detallarán todas las pruebas y simulaciones a las que se verá sometido el sistema para verificar su correcto funcionamiento.
6. **Conclusiones**: Capítulo final de esta memoria, en el que se describen las distintas conclusiones obtenidas durante la realización de este trabajo, así como posibles desarrollos futuros.

2 ESTADO DEL ARTE

Ante la realización de un proyecto de estas características, es fundamental realizar un estudio previo y detallado de las tecnologías y soluciones que existen actualmente en el mercado, de manera que el sistema sea lo más preciso y eficiente posible. Parte de este estudio consiste en contrastar las distintas alternativas que existen para afrontar un mismo problema, valorando las ventajas y desventajas de casa una de ellas y eligiendo la que mejor se adapte a las necesidades del proyecto.

En este apartado se detalla el estándar de comunicación inalámbrica 802.11 y sus alternativas, y los distintos métodos de localización utilizados en WiFi, así como sus alternativas y los motivos por los que se han elegido éstos frente a otros.

2.1 Protocolo IEEE 802.11

2.1.1 Introducción

Una WLAN [4] (Wireless LAN[1]) es una red inalámbrica en la que una serie de dispositivos se comunican entre sí en zonas geográficas limitadas sin necesidad de tendido de cable entre ellos. La gran ventaja de esta tecnología es que ofrece movilidad al usuario y requiere una instalación muy sencilla.

Las redes inalámbricas cumplen con los estándares genéricos aplicables a las LAN cableadas (por ejemplo, IEEE 802.3 o equivalentes).

Entre los componentes que conforman una WLAN podemos mencionar los siguientes:

- Clientes (STA), dotados de una Tarjeta Interfaz de Red (NIC[2]) que incluye un transceptor radio y la antena.
- Puntos de Acceso (Access Points o APs), que permiten enviar la información de la red cableada (por ejemplo, Ethernet) hacia los clientes (STA);
- Controlador de APs que son necesario para despliegues que requieren varios APs por razones de cobertura y/o tráfico.

Este último puede incorporar funcionalidades añadidas, como pueden ser:

- de AP.
- de cliente VPN[3].
- de cliente RADIUS[4] para labores de autentificar y autorizar con un servidor AAA apropiado (Autentificación, Autorización y Accounting),
- de routing, permitiendo encaminar tráfico a otras redes distintas.
- de firewalls[5], facilitándonos una capa añadida de seguridad para nuestra red.

[1] LAN (Local Area Network): red de computadoras que abarca un área reducida, como la de un hogar, un departamento o un edificio.
[2] NIC (Network Interface Card o Network Interface controller): es un componente de hardware que conecta una computadora a una red informática y que posibilita compartir recursos
[3] VPN (Virtual Private Network): es una tecnología de red que permite una extensión segura de la red de área local (LAN) sobre una red pública o no controlada como Internet.
[4] RADIUS (Remote Authentication Dial-In User Service): es un protocolo de autenticación y autorización para aplicaciones de acceso a la red o movilidad IP.
[5] Firewall (cortafuegos): parte de un equipo o red con el objetivo de bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas.

2.1.2 Historia

La compatibilidad entre estos equipos anteriormente explicados, era uno de los grandes problemas que podrían evitar la expansión de las redes inalámbricas. Así pues, la IEEE[6] creó un grupo de trabajo específico para esta tarea de compatibilidades llamado 802.11. [5]

El 802.11 es un estándar que define el uso de las dos capas inferiores de la pila de protocolos OSI (Física y Enlace) con el fin de utilizarlas en una Red de Área Local Inalámbrica en las bandas de frecuencias de 2.4, 5 y 60 GHz. Además, se definen los conceptos básicos para el funcionamiento de estas redes, como por ejemplo las Estaciones Base, el formato de la trama o los requerimientos de potencia.

El problema no se llegó a solventar del todo hasta 1996 con la creación del organismo Wireless Fidelity Alliance (WiFi Alliance), ya que el estándar 802.11 constaba de varios puntos con cierta ambigüedad que podían ser interpretados de distinta manera por los fabricantes provocando incompatibilidades entre equipos.

La WiFi Alliance permitió homogeneizar productos y hacer posible su implantación en el mercado de consumo. A partir de aquí fue cuando estas redes comenzaron a ser usadas por el público en general.

Fue tal la labor que desempeñó la WiFi Alliance, que hoy en día, las redes inalámbricas se conocen como redes WiFi en referencia a este organismo.

El funcionamiento más común de IEEE 802.11 actualmente se localiza en la banda de frecuencias de 2.4GHz. Cuando se trabaja en esta banda, se divide en 14 canales de 22MHz separados 5MHz entre sí. Por lo tanto, existe una clara interferencia entre canales, por lo que se recomienda que solamente se utilicen los canales no interferentes 1, 6 y 11. El funcionamiento en la banda de 5GHz es algo más complicado de explicar, ya que es diferente en función de la legislación de cada país o región.

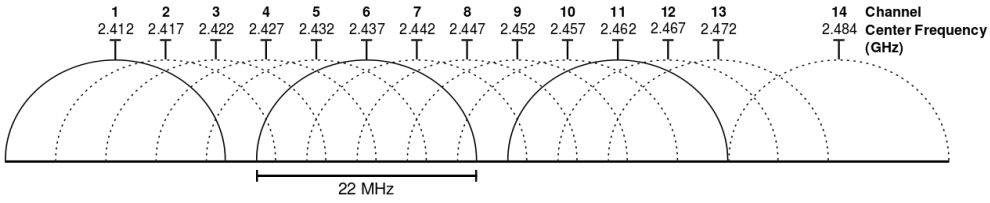


Figura 1. Reparto de canales IEEE 802.11 en la banda de 2.4GHz.

Fuente: repositorio.unican.es/xmlui/handle/10902/8797

[6] IEEE (Institute of Electrical and Electronics Engineers) es una asociación mundial de ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas.

2.1.3 Formato de la trama

Los datos de las capas superiores se encapsulan en una trama IEEE 802.11. Los campos más relevantes que aparecen en la cabecera de estas tramas son:

- Type/Subtype: Especifica el tipo (Control, Gestión o Datos) y subtipo de la trama.
- Address: A diferencia de la trama Ethernet, en el que solo hay direcciones de origen y destino, aparecen 4 direcciones. Esto es porque cabe la posibilidad de realizar forwarding desde, hacia o a través del Sistema de Distribución.
- Sequence Control: Para manejar correctamente retransmisiones.
- Power Management: Indica si las funciones de control de potencia están o no activas.

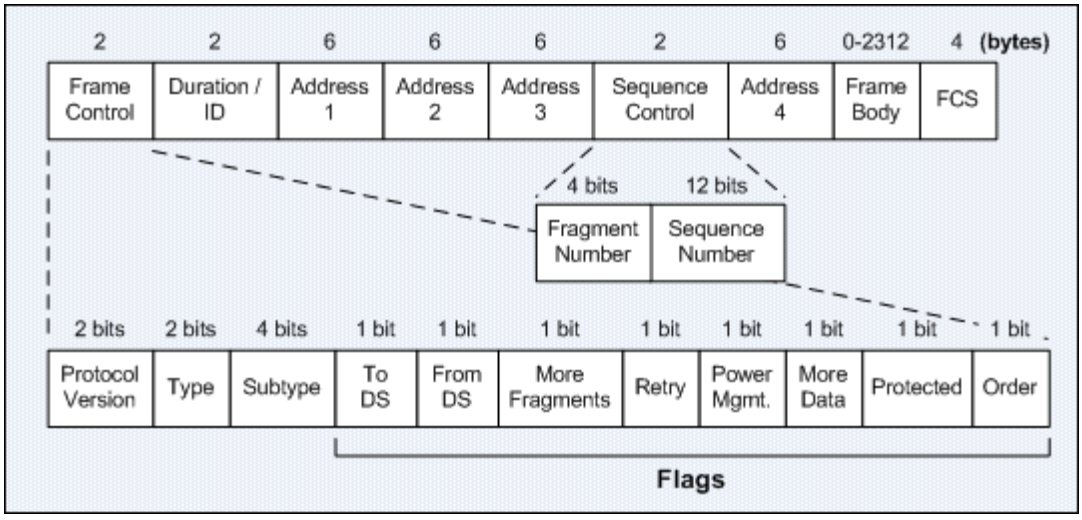


Figura 2. Formato de la trama IEEE 802.11.

Fuente: repositorio.unican.es/xmlui/handle/10902/8797

Gracias al campo de tipo y subtipo de la trama, es posible diferenciar la funcionalidad de cada paquete y, en ocasiones, incluso la procedencia del mismo. Esto, como se verá más adelante, será útil a la hora de filtrar los paquetes que llegan a un dispositivo IEEE 802.11 y dará la posibilidad de procesar los distintos tipos de paquetes de la manera que corresponda en cada caso.

Además de esta estructura de cabecera, es importante destacar un apartado más que será necesario para la consecución del proyecto: la cabecera Radiotap. Esta cabecera, que no se transmite, es añadida por el driver de la tarjeta de red y aporta información de control interesante sobre el medio radio, como por ejemplo el ruido que introduce la antena, la potencia recibida por la tarjeta o la velocidad binaria a la que se transmiten los datos.

2.1.3.1 Paquetes Probe Request

Existen dos maneras de realizar la búsqueda de Puntos de Acceso IEEE 802.11. La primera es la búsqueda pasiva, en la que un dispositivo espera a escuchar los mensajes de tipo Beacon que envían periódicamente los AP. Esta forma es energéticamente costosa, ya que obliga al dispositivo móvil a estar escuchando el canal todo el tiempo para encontrar los paquetes necesarios. El otro método es la búsqueda activa, en la que el dispositivo envía un mensaje al canal y espera ser respondido por los AP.

Los paquetes de tipo 0 y subtipo 4 se denominan Probe Request. Estos paquetes son los que se utilizan en la búsqueda activa. El funcionamiento es el siguiente:

1. El dispositivo que busca conectarse a un AP envía este paquete a toda la red (dirección Broadcast). En este paquete se incluye una lista de los AP a los que desea conectarse, por ejemplo, los que se ha conectado previamente y conoce.
2. Los AP que cumplan las características para que el dispositivo se conecte a ellos le contestan con un mensaje de Probe Response.
3. El dispositivo escoge un AP de la lista de los que le responden y le envía una petición de asociación, con su correspondiente sistema de autenticación si lo hubiera, por ejemplo, IEEE 802.1X.

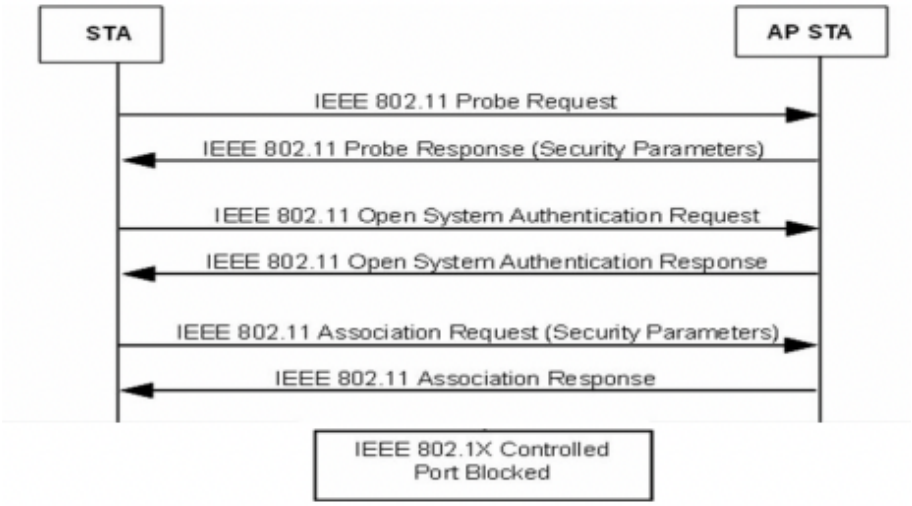


Figura 3. Asociación de un STA a un AP 802.11.

Fuente: repositorio.unican.es/xmlui/handle/10902/8797

En el marco de este trabajo, este tipo de paquetes es relevante dado que es el único formato de paquete que cualquier dispositivo va a enviar independientemente de su estado, es decir, aquellos que no estén necesariamente conectados a una red IEEE 802.11. Además de esto, usar este tipo de paquete nos asegura que el dispositivo que estamos detectando no será un AP, ya que éstos envían Probe Response. Por lo tanto, lo que se aprovechará durante este proyecto será, en principio, detectar este tipo de paquetes en la red y extraer de ellos la información que nos permita localizar al STA que lo ha enviado.

2.1.3.2 Modos de monitorización

Las interfaces de acceso WiFi, más concretamente sus drivers, ofrecen dos maneras de escuchar el canal radio inalámbrico en busca de paquetes IEEE 802.11.

La primera es el denominado modo Promiscuo, en el que un dispositivo se conecta a una red y captura todos los paquetes que circulan por ella, incluso aquellos cuya dirección de destino no es la suya.

El otro modo es el modo Monitor, en el que un dispositivo captura todos los paquetes que aparecen en el entorno IEEE 802.11. Los dos modos son muy similares, pero la mayor diferencia radica en el hecho de que en el modo Promiscuo es necesario estar previamente conectado a la red. Por ello, no es apropiado si lo que se quiere es tener una visión de todos los dispositivos que aparecen en la red global.

2.1.4 Evolución

A lo largo de los años ha ido evolucionando el estándar y se han añadido nuevas funcionalidades y mejoras. Cada vez que aparece una nueva versión, que aporta características nuevas con respecto a las anteriores, se publica con el formato IEEE 802.11 más una letra que la distingue.

A continuación, se detallan los estándares existentes ordenados decrecientemente según su publicación [6].

2.1.4.1 802.11ah

También conocido como WiFi HaLow, 802.11ah define el funcionamiento de redes exentas de licencia en bandas de frecuencia por debajo de 1 GHz (típicamente la banda de 900 MHz), excluyendo las bandas de TV. El propósito de 802.11ah es crear redes WiFi de rango extendido que sean más remotas en el espacio de 2.4GHz y 5GHz, con velocidades de datos de hasta 347Mbps. Además, la norma apunta a tener un menor consumo de energía, útil para que los dispositivos de Internet de las cosas se comuniquen con mucha energía. Pero podría competir con las tecnologías Bluetooth en el hogar debido a sus menores necesidades de energía. El protocolo fue aprobado en septiembre de 2016 y publicado en mayo de 2017.

2.1.4.2 802.11ac

Los enrutadores inalámbricos domésticos actuales son compatibles con 802.1ac y funcionan en el espacio de frecuencia de 5 GHz. Con entrada múltiple, salida múltiple (MIMO[7]), este estándar admite velocidades de datos de hasta 3.46 Gbps. Algunos proveedores incluyen tecnologías que admiten la frecuencia de 2,4 GHz a través de 802.11n, que brindan soporte para dispositivos de clientes más antiguos que pueden tener radios 802.11b/g/n, pero también proporcionan ancho de banda adicional para velocidades de datos mejoradas.

Se desarrolló entre el año 2011 y el 2013, y finalmente aprobada en enero de 2014.

2.1.4.3 802.11ad

Aprobado en diciembre de 2012, 802.11ad es muy rápido: puede proporcionar hasta 6.7 Gbps de velocidad de datos en la frecuencia de 60 GHz, pero eso tiene un coste de distancia de sólo 3.3 metros del punto de acceso.

2.1.4.4 802.11n

El primer estándar para especificar MIMO, 2.4GHz y 5GHz, con un incremento significativo en la velocidad máxima de transmisión de 54 Mbps a un máximo de 600 Mbps. El estándar 802.11n fue ratificado por la organización IEEE el 11 de septiembre de 2009.

2.1.4.5 802.11g

Aprobado en junio de 2003, 802.11g fue el sucesor de 802.11b, capaz de alcanzar velocidades de hasta 54Mbps en la banda de 2.4GHz, igualando la velocidad 802.11a pero dentro del rango de frecuencia más bajo.

2.1.4.6 802.11a

Primer estándar después de la aprobación en junio de 1997 del ya existente 802.11. Está provista para operar en la frecuencia de 5GHz, con velocidades de datos de hasta 54Mbps. Contrariamente, 802.11a salió después de 802.11b, causando cierta confusión en el mercado.

2.1.4.7 802.11b

Lanzado en septiembre de 1999, opera en la frecuencia de 2.4 GHz y proporciona hasta 11 Mbps. Curiosamente, los productos 802.11b llegan al mercado antes que 802.11a, que fue aprobado al mismo tiempo, pero no llegó al mercado hasta más tarde.

[7] MIMO (Multiple Input Multiple Output). Un router MIMO (802.11n) es aquel permite una cobertura mayor en zonas de difícil acceso eliminando en lo posible la pérdida de paquetes de datos vía inalámbrica, también nos proporciona mayor velocidad inalámbrica por usar varias antenas de forma simultánea.

2.1.4.8 802.11

El primer estándar (1997), que proporciona una velocidad de datos de hasta 2 Mbps en la frecuencia de 2,4 GHz. Tiene un alcance de 20 metros de interior.

2.1.4.9 Comparación

Para resumir, se muestra una tabla en la que se detallan los aspectos más importantes de las diferentes tecnologías inalámbricas.

Estándar	Velocidad máxima	Frecuencia	Compatibilidad con versiones anteriores
802.11a	54 Mbps	5 GHz	No
802.11b	11 Mbps	2,4 GHz	No
802.11g	54 Mbps	2,4 GHz	802.11b
802.11n	600 Mbps	2,4 GHz o 5 GHz	802.11b/g
802.11ac	1,3 Gbps (1300 Mbps)	2,4 GHz y 5,5 GHz	802.11b/g/n
802.11ad	7 Gbps (7000 Mbps)	2,4 GHz, 5 GHz y 60 GHz	802.11b/g/n/ac

Figura 4. Comparación de los estándares 802.11 más relevantes.

Fuente: www.top10games.es

2.1.5 Futuros estándares

Existen una serie de tecnologías que están siendo desarrolladas, están pendientes de ser aprobadas y desplegadas, o bien no han sido aprobadas según se esperaba. Éstas son las siguientes. [7]

2.1.5.1 802.11aj

También conocido como China Millimeter Wave, este estándar define las modificaciones posteriores en el 802.11ad físico y la capa MAC para permitir el funcionamiento en la banda de frecuencia de 59-64 GHz de China. El objetivo es mantener la compatibilidad con el estándar 802.11ad (60GHz) cuando opera en ese rango de 59-64 GHz y operar en la banda China de 45GHz, mientras se mantiene la experiencia del usuario 802.11. La aprobación final se esperaba en noviembre del 2017.

2.1.5.2 802.11ak

Existen algunos productos en los campos del entretenimiento doméstico y el control industrial que tienen una capacidad inalámbrica de 802.11 y función Ethernet 802.3. El objetivo de esta norma es ayudar a los medios 802.11 a proporcionar conexiones internas como enlaces de tránsito dentro de las redes 802.1q asociadas, especialmente en los campos de velocidad de datos, seguridad estandarizada y mejoras de calidad de servicio. La aprobación estaba prevista para noviembre del 2017.

2.1.5.3 802.11ax

Conocido como WLAN de alta eficiencia, el 802.11ax tiene como objetivo mejorar el rendimiento en implementaciones de WLAN en locaciones densas, como estadios deportivos y aeropuertos, mientras sigue operando en el espectro de 2,4GHz y 5GHz. El grupo tiene como objetivo una mejora cuádruple en el

rendimiento en comparación con los estándares 802.11n y 802.11ac., a través de la utilización del espectro de uso más eficiente. Actualmente, su aprobación está estimada para julio del 2019.

2.1.5.4 802.11ay

También conocido como Next Generation 60GHz, el objetivo de este estándar es admitir un rendimiento máximo de al menos 20Gbps dentro de la frecuencia de 60GHz (802.11ad logra actualmente hasta 7Gbps), así como aumentar el alcance y la confiabilidad. Se espera que el estándar sea aprobado entre septiembre y noviembre del 2019.

2.1.5.5 802.11az

Llamado Next Generation Positioning (NGP), se formó un grupo de estudio en enero del 2015 para abordar las necesidades de una "estación para identificar su posición absoluta y relativa a otra estación o estaciones a las que está asociada o no asociada". Los objetivos del grupo serían definir modificaciones en las capas MAC y PHY que permitan: "Determinar la posición absoluta y relativa con mayor precisión en relación con el protocolo Fine Timing Measurement (FTM) que se ejecuta en el mismo tipo PHY, reducir simultáneamente el uso de medios inalámbricos y el consumo de energía existentes, y lograr que sean capaces de aumentar la escala a implementaciones densas". Actualmente, se estima que la aprobación de esta norma se llevará a cabo en marzo del 2021.

2.1.5.6 802.11ba

Conocido también como "Wake-Up Radio" (WUR), involucra una nueva tecnología destinada a extender la duración de la batería de los dispositivos y sensores dentro de una red de Internet de las cosas. El objetivo de WUR es "reducir en gran medida la necesidad de recargar y reemplazar baterías con frecuencia mientras se mantiene el rendimiento óptimo del dispositivo". Actualmente, se espera que se apruebe en julio del 2020.

El rango de frecuencias usado por el estándar es no licenciado, lo que significa que no es necesario poseer una licencia para transmitir en esas frecuencias. Esto no ocurre así, por ejemplo, en 3G o GPRS, que si se necesita una licencia especial para poder realizar transmisiones en las frecuencias correspondientes a estas tecnologías.

2.1.6 Alternativas a 802.11

2.1.6.1 Bluetooth

La tecnología Bluetooth se considera un competidor de la tecnología WiFi en los sistemas de posicionamiento en interiores, en particular, desde la adopción generalizada del Bluetooth de Baja Energía (Bluetooth Low Energy o BLE), debido a que es compatible con la mayoría de los smartphones modernos. Es una tecnología de bajo coste, y a su reducido consumo de energía, lo que permite a los emisores funcionar con pilas durante varios meses o incluso años.

Al igual que en los sistemas WiFi, se utiliza la RSSI para obtener la localización. En estos sistemas, existen balizas fijas denominadas beacons (análogas a los puntos WiFi) que emiten la señal. La señal es recogida por un STA que lleva el usuario y se aplica la técnica correspondiente para calcular la distancia que se verá en el siguiente apartado.

Figura 5. Balizas beacon

Fuente: repositorio.unican.es/xmlui/handle/10902/8797

2.1.6.2 RFID

Un sistema RFID consiste en lectores y etiquetas RFID. El lector puede obtener los datos emitidos por las etiquetas. Estas últimas actúan como una antena, siendo capaces de emitir señales de información, principalmente su ID único. Las etiquetas se pueden clasificar en dos grupos en función de cómo obtienen la energía para funcionar:

- Pasivas si utilizan la pequeña energía emitida por el lector, recogida mediante la pequeña antena;
- Activas si tienen su propia fuente de alimentación y transmiten periódicamente su ID.

Esta tecnología tiene múltiples usos, como el control de acceso de personal a ciertas áreas. También se han utilizado sistemas RFID para la localización, especialmente cuando la localización precisa del usuario no necesita ser conocida en todo momento, sino sólo cuando pasa por lugares de control importantes. En estos casos, la ubicación del usuario se da a menudo en forma de ubicación lógica (ejemplos: “en la puerta”, “dentro de la sala”, “fuera del recinto”) y no en un sistema de coordenadas.

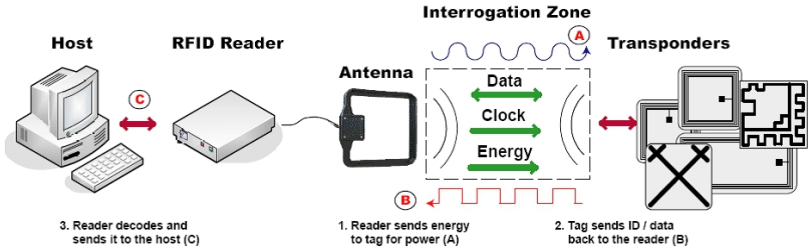


Figura 6. Componentes de un sistema RFID

Fuente: repositorio.unican.es/xmlui/handle/10902/8797

2.1.6.3 Campo magnético

La mayoría de estos sistemas utilizan la intensidad del campo magnético de la Tierra y su orientación para realizar el posicionamiento.

Un IPS[8] basado en campos magnéticos utiliza un magnetómetro para medir las variaciones del campo magnético, que se utilizará para posicionar al usuario. La estimación de la posición se realiza comúnmente a través de técnicas como el fingerprinting, el cual se detalla en apartados posteriores.

2.1.6.4 Sensores Inerciales

Suele denominarse navegación por estima. Es la navegación por medios analíticos teniendo en cuenta la situación inicial del usuario, su rumbo y su velocidad. El problema principal de este tipo de navegación es la acumulación progresiva de errores, ya que un pequeño error en la dirección del usuario podría significar un error enorme cuando se viaja una distancia muy grande.

Este tipo de sistemas utilizan acelerómetros y giroscopios, combinando la información con otros sensores para lograr un buen rendimiento. Los acelerómetros pueden utilizarse para determinar las variaciones en la posición del usuario cuando se detecta aceleración en una determinada dirección.

Tal estimación puede mejorarse usando un giroscopio para detectar los cambios de dirección. Por otro lado, mediante el acelerómetro también podemos confirmar el hecho de que el usuario esté caminando, reconociendo el movimiento que se produce al caminar (podómetro).

[8] IPS (Indoor Positioning System): Sistema de Posicionamiento en Interiores.

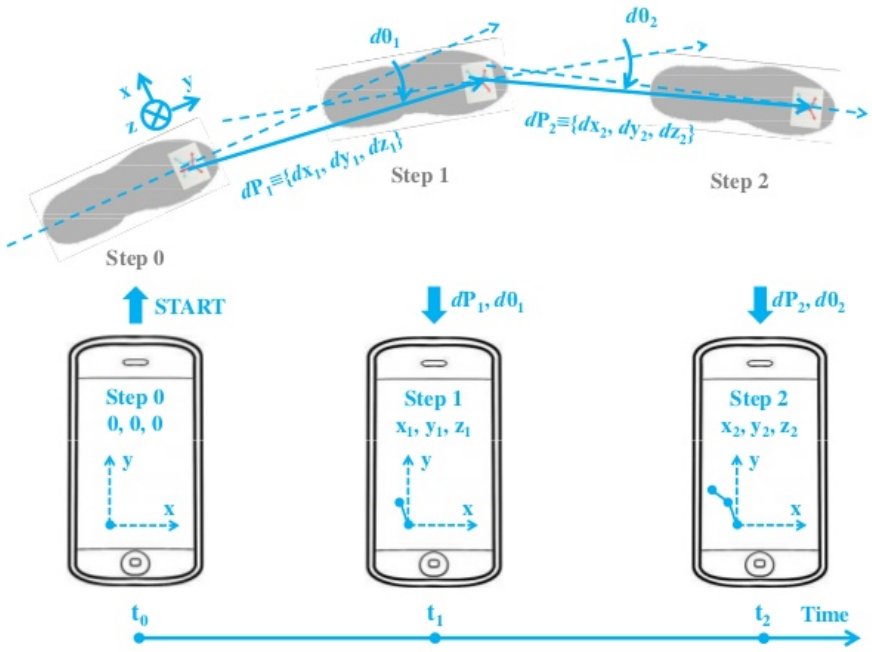


Figura 7. Navegación por estima

Fuente: repositorio.unican.es/xmlui/handle/10902/8797

2.1.6.5 Visión por computador

Estos sistemas hacen uso de la información recogida por cámaras y técnicas de procesamiento de imagen para identificar y rastrear objetos. Hay dos enfoques para estos sistemas.

En un primer enfoque, una o varias cámaras se fijan en el entorno en el que se supervisarán a los usuarios y se utilizan estas para rastrearlos.

En otro enfoque menos común, es el usuario el que lleva la cámara, que va capturando imágenes o vídeos, que se comparan con archivos previamente almacenados en la base de datos.

2.1.6.6 Infrarrojos

Un sistema de infrarrojos (IR, Infrared) está compuesto de un diodo emisor de luz infrarroja, que emite una señal de ráfagas de luz no invisible, y un fotodiodo receptor para detectar y capturar los pulsos de luz, que son después procesados para recuperar la información.

La fiabilidad de estos sistemas se ve afectada por muchas características de la señal óptica emitida, tales como su directividad (en qué grado es unidireccional), así como su forma de reaccionar ante obstáculos. La mayoría de sistemas IR requieren una línea de visión clara entre emisor y receptor.

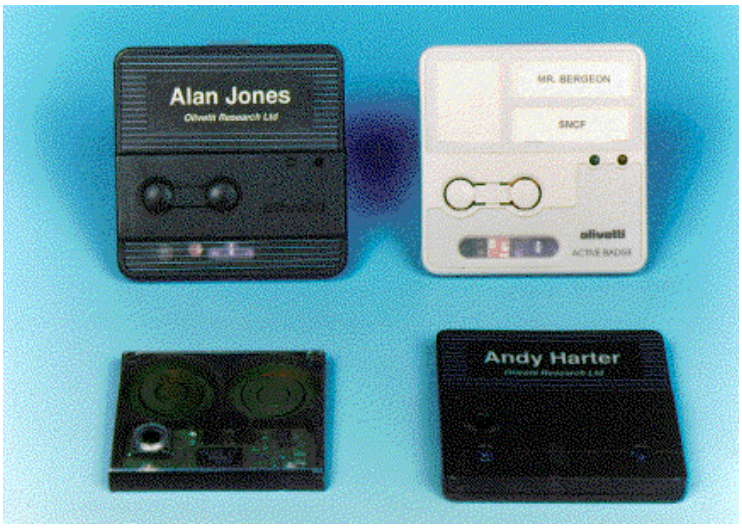


Figura 8. Sensores IR

Fuente: www.cl.cam.ac.uk/research/dtg/attarchive/ab.html

2.1.6.7 VLC

La comunicación con luz visible (VLC, Visible Light Communication) es una tecnología que usa luz visible para transmitir datos. Se puede utilizar cualquier tipo de lámpara, pero las luces LED son las más apropiadas. La transmisión de datos usando luz visible es posible debido a la capacidad de la fuente de luz de encenderse y apagarse en muy cortos intervalos de tiempo. Este parpadeo puede ser tan rápido que no pueda ser percibido por el ojo humano.

El principio detrás de VLC es que cada una de las lámparas fijas tiene una codificación de parpadeo diferente, por lo que el sensor, que podría ser transportado por el usuario, recibe la luz y compara la modulación con los esquemas de codificación conocidos y determina cuál es el dominante, lo que permite asociar la posición del sensor con la proximidad de la correspondiente lámpara.

Las ventajas de estos sistemas es que permite la reutilización de la infraestructura de luz artificial disponible; así como que tal disposición no es intrusiva, porque los usuarios humanos solo ven lámparas ordinarias fijadas en el techo.

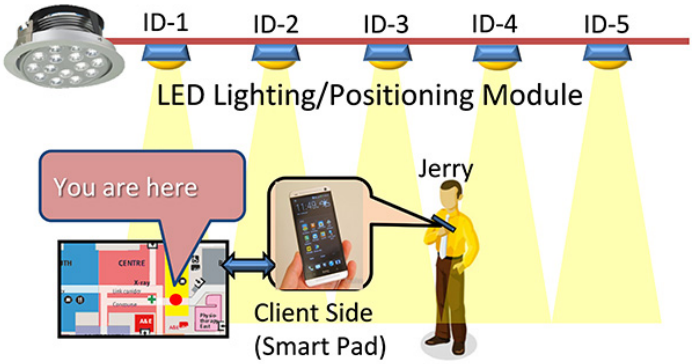


Figura 9. Sistema de posicionamiento en interiores mediante VLC

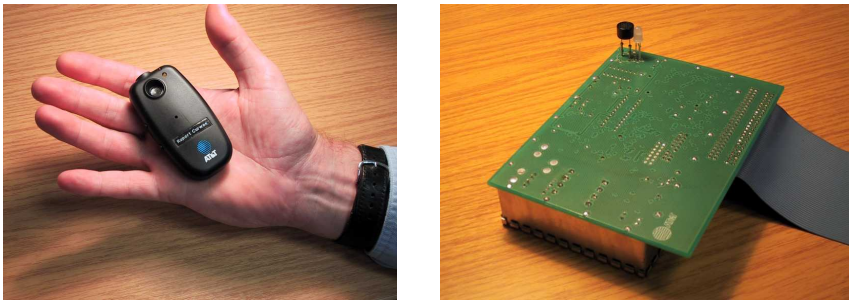
Fuente:
www.itri.org.tw/eng/DM/PublicationsPeriods/6212731510633560 2/content/focus2.html

2.1.6.8 Sonido audible

También es posible utilizar este tipo de señal sonora para codificar la información para sistemas de localización. Es evidente que la idea de usar sonido audible tiene demasiados inconvenientes; el principal, que molestaría a las personas cercanas. No obstante, existen proyectos sofisticados en los que se utilizan fuentes de sonidos ya presentes, como la música en centros comerciales y otros lugares públicos.

2.1.6.9 Ultrasonido

Los sistemas de localización por ultrasonidos utilizan frecuencias de sonido superiores al rango audible para determinar la posición del usuario, utilizando para ello el tiempo que tarda una señal ultrasónica en desplazarse de un transmisor a un receptor. Una ventaja evidente de las señales de ultrasonidos frente a las señales audibles es que las primeras no son detectables por los seres humanos.



(a) Emisor de ultrasonidos. (b) Dispositivo sensor

Figura 10. Sensores de ultrasonidos

Fuente: www.cl.cam.ac.uk/research/dtg/attarchive/bat

2.1.6.10 Ultra-WideBand (UWB)

El término Ultrawideband (UWB) [8] se utiliza para hacer referencia a cualquier tecnología de radio que usa un ancho de banda mayor de 500 MHz o del 25% de la frecuencia central. La transmisión de pulsos muy cortos hace que esta tecnología sea adecuada para aplicaciones tipo radar para zonas muy localizadas obteniéndose rastreos de gran resolución, y una precisión muy alta en aplicaciones de localización y detección; pero es su enorme ancho de banda y, sobre todo, el menor coste y nivel de complejidad de los sistemas que se basan en ella, lo que hace que la tecnología UWB, se haya perfilado recientemente como una de las tecnologías emergentes sobre las que se podría apoyar la próxima generación de sistemas inalámbricos.

Un aspecto negativo de UWB es el alcance, ya que, si se aumenta éste, ha de ser a costa de disminuir la velocidad de transmisión, debido a las limitaciones de potencia. El alcance también se ve afectado por la presencia de obstáculos que tiendan a reflejar las señales, aunque también es cierto que la capacidad de UWB para atravesar estructuras u objetos es mucho mayor que la de otras tecnologías inalámbricas. Además, acualmente esta tecnología tiene el inconveniente de requerir infraestructura propia.

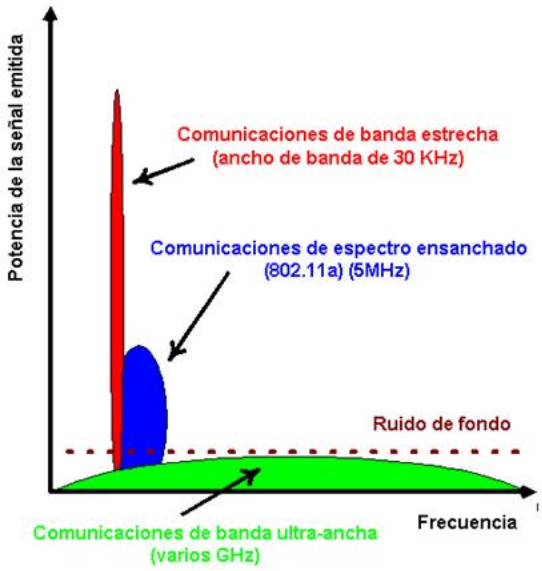


Figura 11. Espectro de frecuencias UWB [8].

2.1.7 Comparación

Para tener una visión un poco más clara, se comparará las diferentes tecnologías inalámbricas analizadas anteriormente mediante el uso de la siguiente tabla, donde CI es el Coste de Instalación y CU el Coste para el Usuario:

Tecnología	Exactitud aprox.	CI	CU	Ventajas	Inconvenientes
Infrarrojos	57cm-7.3m	+	-	Barato para el usuario	Interferencia de la luz del sol
VLC	10cm	+	-	Barato para el usuario	Infraestructura cara
Ultrasonidos	1cm-2m	+	+	Buena precisión	Coste, interferencias
Sonido audible	Metros	-	-	Bajo coste	Baja precisión
Bluetooth	30cm-metros	-	-	Bajo coste, buena precisión	El rango de las balizas es reducido
RFID	1-5m	+	-	Coste bajo	Precisión muy reducida
Campo magnético	2m	-	-	No requiere infraestructura, buena precisión	Requiere mapeo
Sensores inerciales	2m	-	-	Coste bajo	Acumula error
Visión por computador	1cm-1m	-	-	Coste bajo	Sensibilidad a cambios de luz
UWB	20cm	+	-	Buena precisión	Poco alcance, Infraestructura propia
WiFi	1.5m	-	-	Bajo coste, buena precisión	Pueden cambiar los APs

Tabla 1. Comparación de tecnologías inalámbricas

Por otro lado, es importante destacar también cual es la cobertura que ofrecen cada una de estas tecnologías. En un estudio realizado en 2014 por Luca Mainetti [9], podemos apreciar esta característica de algunas de las tecnologías estudiadas:

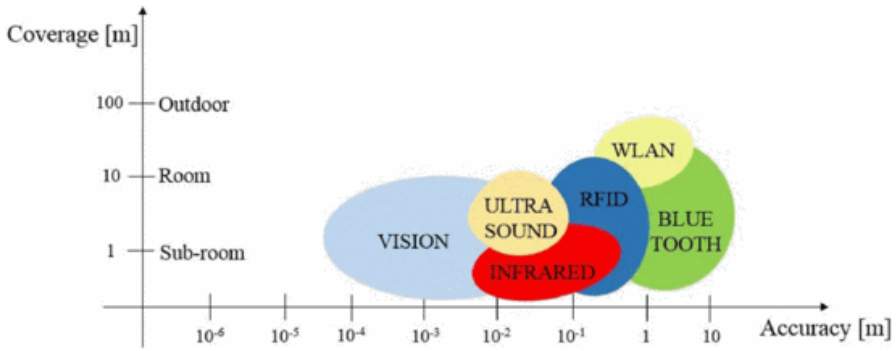


Figura 12. Tecnologías comparadas por su exactitud y cobertura [9].

2.1.8 Decisiones

En este punto, se van a detallar las decisiones tomadas para la elección y descartes de cada tecnología.

En relación al Infrarrojo, además de que supone un coste elevado de instalación, la mayoría de sistemas IR requieren una línea de visión clara entre emisor y receptor. Como es evidente, este requisito en sistemas IR IPS es una gran desventaja, ya que no habrá detección en aquellas zonas ocluidas del transmisor o el sensor. Es por estas razones por las que el **Infrarrojo es descartado**.

Todas aquellas tecnologías que requieran un coste elevado ya sea al usuario, o de instalación o infraestructuras, las descartaremos también. Éstas son: **VLC y Ultrasonidos**.

De igual manera, **descartamos RFID** puesto que tiene una baja precisión además de que requiere un coste de instalación considerable.

Aunque **Bluetooth** nos permite obtener una de las mejores precisiones, el rango en el cual nos permite intercambiar información, es decir, colocar las balizas, es reducido. Por esta razón también es descartado.

El método de **Sonido audible**, aunque tiene bajo coste tanto para el usuario como para el ISP, la precisión es bastante pobre y su sonido puede ser muy molesto en determinadas situaciones, lo que lleva a su descarte.

En relación al Campo magnético, su coste es bajo, pues usamos el propio campo de la Tierra. El problema surge del hecho de mapear la zona, que, en nuestro caso está destinada para localizar. Cuánta más precisión necesitemos, más tediosa se volverá la tarea de mapeo ya que tendremos que medir más minuciosamente el área. Esta es la razón por la que el **Campo magnético es descartado**.

Como en todos los casos, hay que tener en cuenta los errores, y más aún cuando estos se multiplican o se suman tras su ocurrencia. Es el caso de los **Sensores inerciales**, cuya tecnología hace que los errores se aumenten linealmente tras una ocurrencia del mismo. Por ello este método es descartado.

En relación a **Visión por computador**, debido a su sensibilidad a los cambios de luz, esta tecnología no nos interesa puesto que no es aplicable siempre.

Aunque la localización por **UWB** nos ofrece muy buena precisión, se necesita de una infraestructura específica, y además el alcance se ve gravemente afectado por obstáculos y limitaciones de potencia.

Por último, nos queda el estándar 802.11 (WiFi). Tiene bajos costes de instalación y de usuario. La precisión

es aceptable. Es cierto que los APs pueden cambiar, pero no debemos olvidar que éstos no cambian en comportamiento por sí solos, sino que, somos nosotros quienes con, una actualización de su software o hardware, los que provocamos esos cambios. La posición de los mismos también es fija, y si es cambiada, lo es conscientemente, salvo casos extremos.

Además, teniendo en cuenta que la degradación de los componentes, que también es aplicable a la gran mayoría de tecnologías anteriores, **WiFi es la tecnología elegida para este proyecto**.

2.2 Métodos de localización

2.2.1 Distancia según RSSI

La manera más directa de estimar la distancia desde una fuente electromagnética a un receptor es estimarla desde un modelo teórico basado en RSSI[9]. Sin embargo, la potencia recibida en un canal 802.11 es un valor que varía mucho dependiendo de factores como el multitrayecto o efectos de desvanecimientos.

Se trata, por tanto, de un método fácil de implementar, pero obtiene una precisión de uno 5-7 metros.

Según la precisión deseada, para obtener una medida estimada de la distancia a partir del RSSI, obtendremos ecuaciones más o menos complejas. La que se ha utilizado en este proyecto, la podemos encontrar en "*A Distance Measurement Wireless Localization Correction Algorithm Based On RSSI*" [10]:

$$RSSI_{dBm} = A - 10klog(d) \qquad [1]$$

$$d = 10^{\frac{A-RSSI}{10n}} \qquad [2]$$

donde:

d = distancia entre transmisor y receptor en metros.

A = potencia en dBm recibida a 1 metro del transmisor.

n = exponente de pérdidas en el trayecto. Para localización en interiores, este parámetro varía de 1.6 a 3.5 aproximadamente.

$RSSI_{dBm}$ = intensidad de la señal recibida en dBm.

[9] RSSI (Received Signal Strength Indicator) es una escala de referencia (en relación a 1 mW) para medir el nivel de potencia de las señales recibidas por un dispositivo en las redes inalámbricas

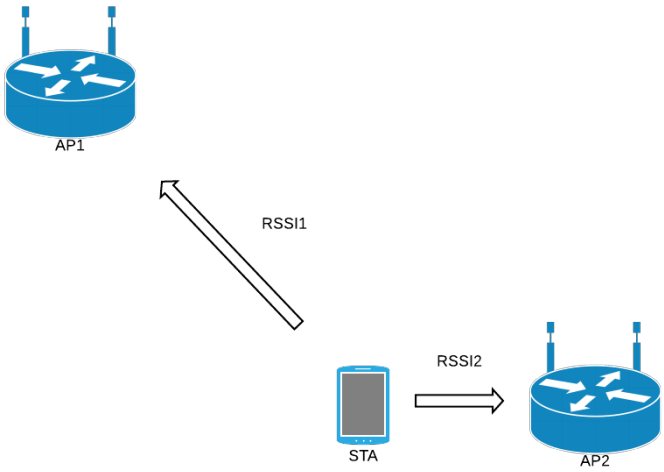


Figura 13. Envío de RSSI a los APs.

2.2.2 Tiempo de vuelo (ToF)

Como las ondas de radio se propagan a la velocidad de la luz, podemos calcular la distancia computando el tiempo transcurrido entre el envío de datos y la recepción de su correspondiente ACK. Un inconveniente existente es el hecho de que los chips de los routers actuales son sobrecargados cuando se les solicitan marcas de tiempos del orden de nanosegundos. Además, hay un lapso de tiempo en el que la trama es procesada y depende principalmente del procesador del equipo. Aun así, este método es capaz de calcular posiciones en un rango de 3 a 5 metros.

A veces también es denominado tiempo de llegada (ToA), y es el tiempo que tarda la señal en llegar del emisor al receptor. Si el receptor obtiene este ToF, entonces puede estimar la distancia la siguiente ecuación:

$$d = \frac{c}{\frac{t}{2}} \quad [3]$$

donde:

d = distancia entre transmisor y receptor.

c = velocidad de la luz en m/s.

t = tiempo de vuelo. Se divide entre dos para no tener en cuenta ida y vuelta (RTT).

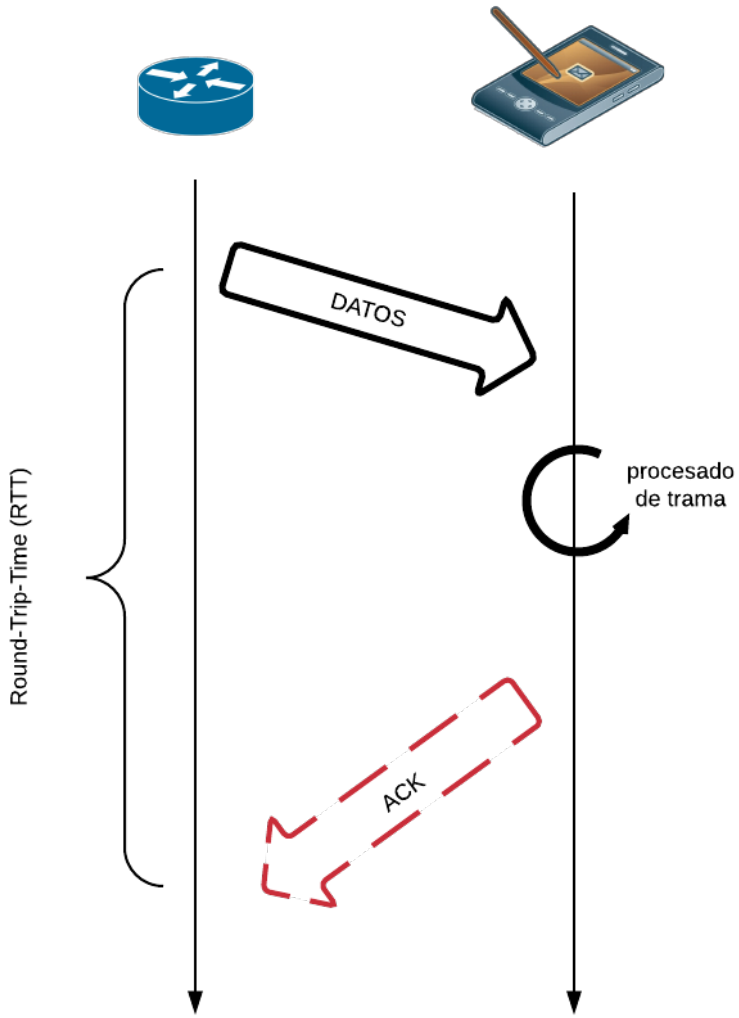


Figura 14. RTT entre un AP y un STA.

2.2.3 Localización por pesos

Este método consiste en asignar pesos a los diferentes puntos de accesos de manera que, aquellos que estén más cerca, les corresponda peso mayor que los que están más lejanos.

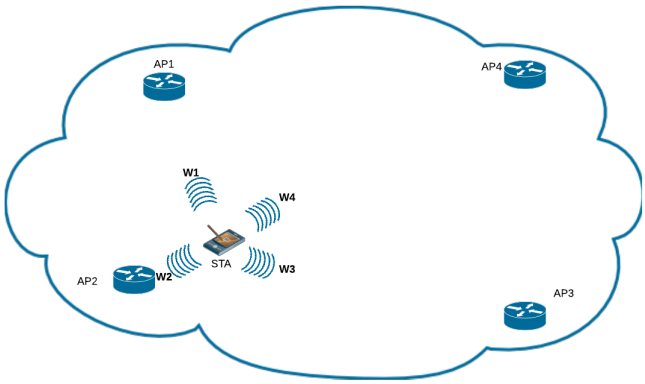


Figura 15. Asignación de pesos para cada AP según su cercanía.

Donde W_i son los pesos asignados al AP_i. De esta manera $W_2>W_1>W_3>W_4$.

La manera de asignar estos pesos se basa en el RSSI recibido, de forma que, a mayor RSSI, mayor es el peso asignado.

En el apartado de simulaciones se discutirá qué fórmulas arrojan mejores pesos.

2.2.4 Diferencia en el tiempo de llegada (DToA)

El principal inconveniente de la computación de tiempos es la incertidumbre que existe en relación al tiempo de computación de las tramas. Sin embargo, podemos suponer que este tiempo es constante, pudiendo así computar la diferencia entre tiempos de llegada entre dos o más APs al mismo STA. Este método requiere una precisión mayor, pero devuelve unos errores de localización entre 1 y 3 metros. Además, es sensible al multitrayecto, paredes y objetos moviéndose, los cuales son también principales inconvenientes del método de localización basado en RSSI.

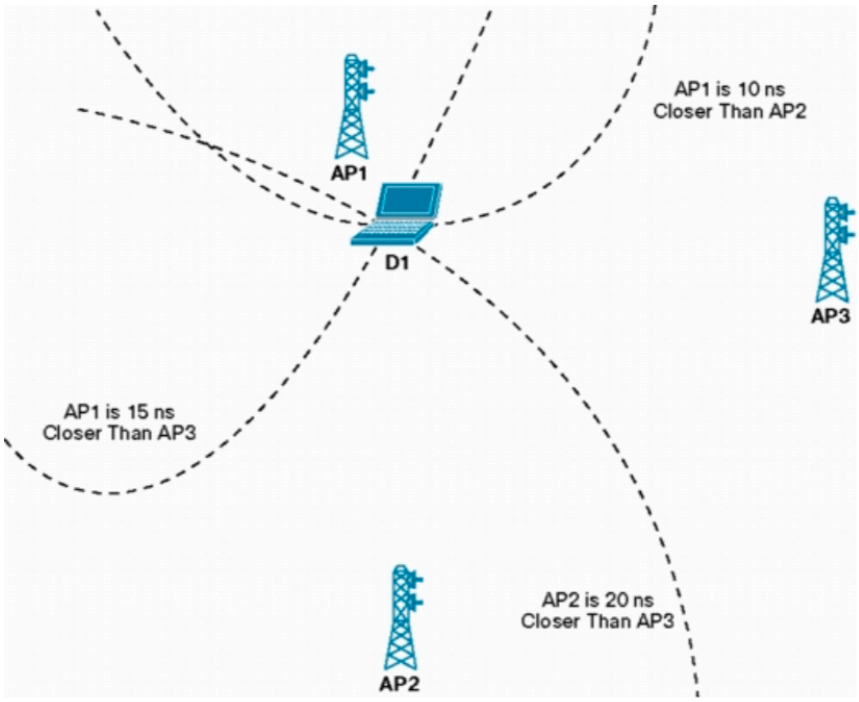


Figura 16. Ejemplo de diferencia en el tiempo de llegada

Fuente: www.cisco.com/c/en/us/solutions/collateral/borderless-networks/context-aware-mobility-solution/white_paper_c11-476796.html

2.2.5 Ángulo de llegada (AoA)

Este método estima la posición del objetivo determinando el ángulo de incidencia en el cual las señales llegan al receptor. El dispositivo emisor define una línea que parte del mismo y que incide en el receptor (objetivo) con un ángulo dado. La combinación de varias líneas de varios dispositivos emisores permiten situar al receptor. Al menos son necesarios dos APs y dos ángulos. Para este método son necesarios puntos de accesos MIMO.

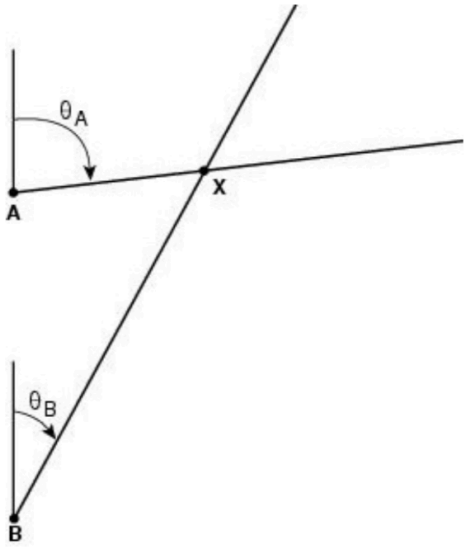


Figura 17. Ángulo de llegada para dos APs

Fuente: www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Mobility/WiFiLBS-DG/wifich2.html

2.2.6 CSI (Channel State Information)

El CSI es una matriz compleja disponible en algunos dispositivos WiFi modernos, el cual revela la naturaleza de las rutas de propagación y cada una de las antenas receptoras. Esto obliga a que el AP sea de tipo MIMO, y cuanta más antenas, más preciso es y más carga computacional requiere.

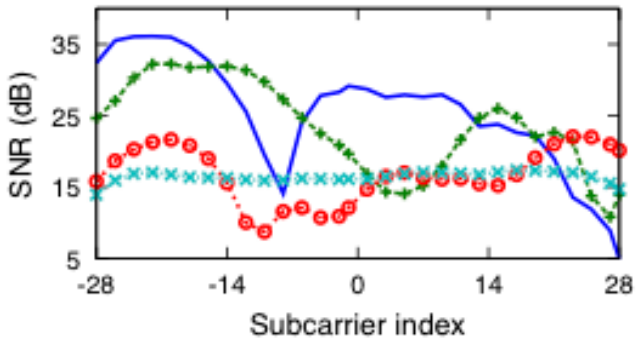


Figura 18. CSI para 4 enlaces SISO[10].

Fuente: https://dhalperi.github.io/linux-80211n-csitool/

[10] SISO (Simple Input Simple Output): se trata de un router que no permite utilizar varias antenas de forma simultánea. Si no es un router MIMO, es SISO.

2.2.7 Fingerprinting

Este obtiene la localización aproximada a partir de mediciones de distintas señales: WiFi, Bluetooth, campo magnético, etc. Se divide en dos fases: entrenamiento y estimación de la posición.

En la fase de entrenamiento, se obtiene un mapa con las intensidades de señal recibidas en distintos puntos, medidas manualmente.

A continuación, para estimar la posición, las intensidades de señal observadas en el dispositivo del usuario se comparan con el mapa construido anteriormente utilizando algoritmos de proximidad, como el de k-vecinos más cercanos, con el objetivo de estimar la posición del usuario.

Estos métodos no funcionan correctamente cuando hay obstáculos que cambian sus posiciones o no han sido tenidos en cuenta.

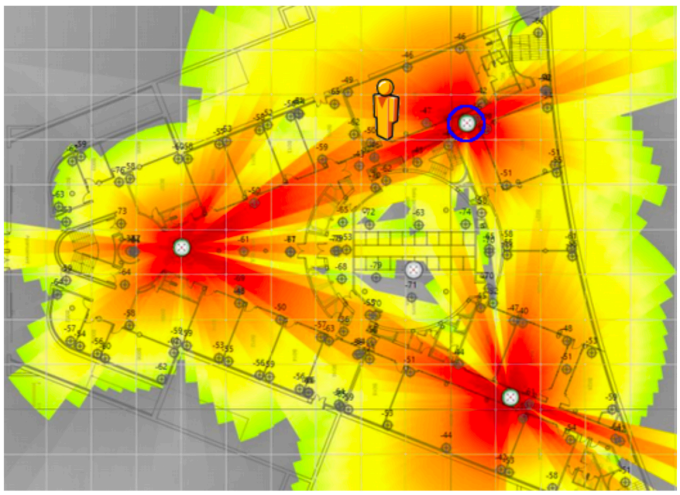


Figura 19. Ejemplo de técnicas de Fingerprinting

Fuente: http://air-go.es/technology/fingerprinting/

En la figura anterior podemos ver un posible mapa construido a partir de las intensidades de señal recibidas de tres puntos de acceso WiFi (se redondea en azul el punto más cercano al usuario). Almacenar esta información en una base de datos sería tan sencillo como crear una tabla en la que las filas se corresponde con un vector de características con la información de las intensidades de señal.

A la hora de localizar a un usuario, tendríamos que obtener las intensidades recibidas de los puntos de acceso, obteniendo otro vector de características. Este vector de características lo compararíamos con todos los vectores almacenados en la base de datos, obteniendo la localización aproximada del usuario.

2.2.8 Calibración

La calibración no es un método de localización como tal, sino un procedimiento capaz de calcular los parámetros desconocidos de la ecuación [2]. Este método, el cual utiliza la técnica de localización basada en RSSI, consiste posicionar un STA entre dos o más APs de manera que haya la menor cantidad de obstáculos entre ellos y su conexión sea lo más directa posible.

Como conocemos la posición real del STA, aplicamos la ecuación [2], hasta encontrar los valores de A y n que hagan que la diferencia entre la distancia estimada en la ecuación anterior y la real sea mínima.

Figura 20: Calibración en una posición libre de obstáculos.

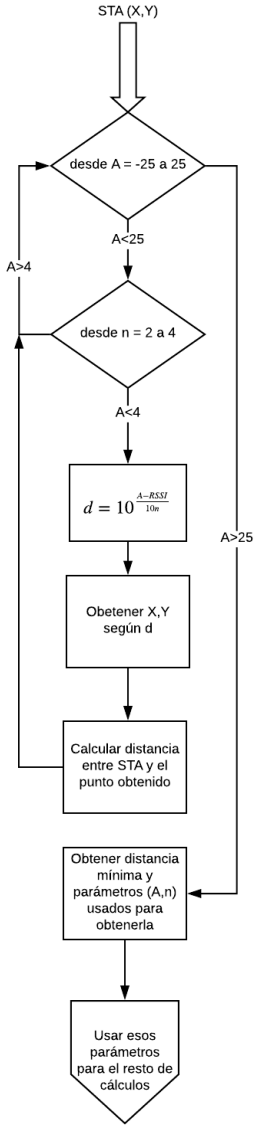


Figura 21: Algoritmo de calibración en una posición libre de obstáculos.

2.2.9 Comparación

En este apartado se comparará los diferentes métodos expuesto anteriormente.

Métodos	RSSI	ToF	TDoA	CSI	AoA	Pesos
Precisión	**5-7 m**	**3-5 m**	**1-3 m**	**0.5-1 m**	**2-4 m**	**~***
Complejidad	**Baja**	**Baja**	**Media**	**Alta**	**Media**	**~***
Degrada con multitrayecto	**Sí**	**No**	**No**	**No**	**No**	**Sí**
Necesita MIMO	**No**	**No**	**No**	**Sí**	**Sí**	**No**
Requiere marcas de tiempo precisas	**No**	**Sí**	**Sí**	**No**	**No**	**No**

Tabla 2. Comparación de métodos de localización

* Dependiendo de la complejidad de las ecuaciones, obtendremos una precisión mejor o peor.

2.3 Método de posicionamiento

La gran mayoría de los métodos de localización devuelven la distancia existente entre los APs y el STA. Por tanto, necesitamos de algún algoritmo capaz de devolver la posición del dispositivo deseado, a partir de las distancias y posiciones de otros. El método seguido en este proyecto es el de la trilateración:

2.3.1 Método de trilateración

La trilateración consiste en determinar la ubicación de un punto fijo por la geometría de esferas, círculos o triángulos [11].

El algoritmo de trilateración empleado en este proyecto es una versión modificada del realizado por Noomrevlis [12].

Su procedimiento se puede resumir como se muestra en las siguientes figuras:

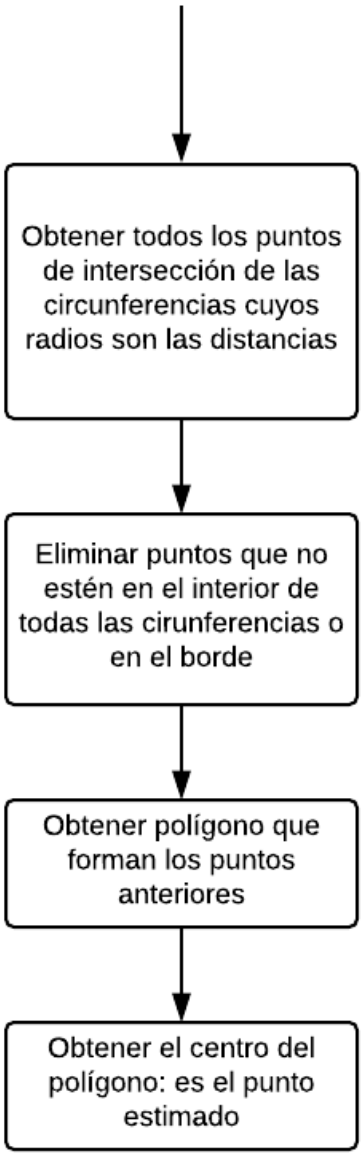


Figura 22: Algoritmo de trilateración.

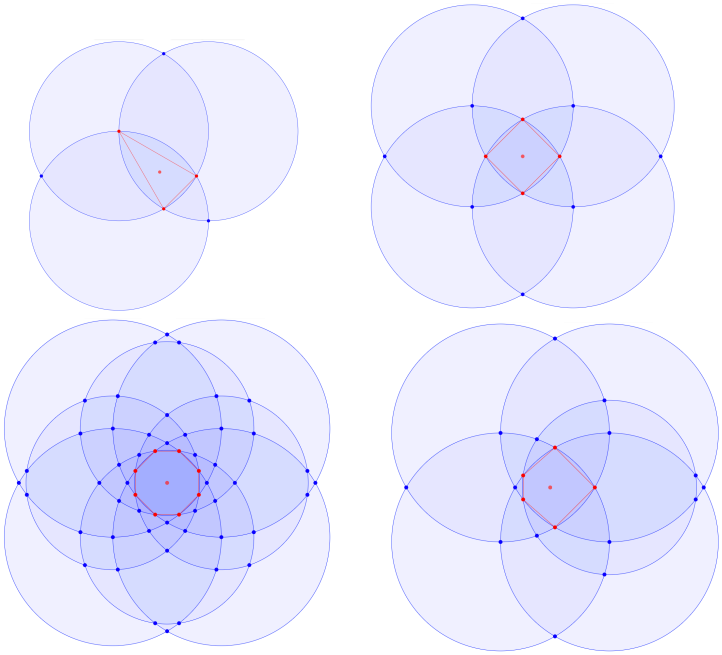


Figura 23. Varios ejemplos de Trilateración.

2.3.2 Alternativas

El método de trilateración no es el único capaz de determinar posiciones. Existen varias alternativas que, a través de diferentes parámetros también calcula posiciones. Éstas son las siguientes.

2.3.2.1 Triangulación

La triangulación consiste en el uso de la trigonometría de triángulos para determinar posiciones de puntos, medidas de distancias o áreas de figuras.

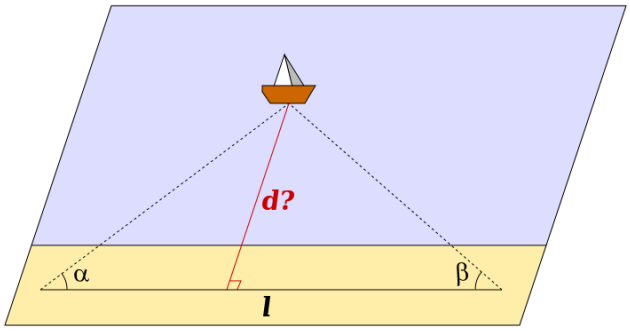


Figura 24. Triangulación

Fuente: es.wikipedia.org

Puesto que necesitamos datos de los que no disponemos, como pueden ser los ángulos, descartamos este método.

2.3.2.2 Multilateración

Al igual que en el método de triangulación, para la multilateración [13] se necesitan las distancias a un número determinado de balizas para calcular la posición de un nodo, solo que, en este caso, las distancias se determinan utilizando TDoA.

También se descarta la multilateración puesto que se necesitan distancias a puntos fijos y usa TDoA para el cálculo de las distancias, lo que conlleva un mayor uso de recursos y una complejidad mayor.

Esta técnica se suele utilizar en el posicionamiento aéreo.

2.3.2.3 Análisis de escena

La técnica de análisis de la escena [13] utiliza las características de una escena observada desde un punto de vista en particular, para extraer conclusiones sobre la posición relativa de los objetos. Habitualmente las escenas observadas se simplifican para obtener características que son fáciles de representar y comparar.

Cuando se produce una diferencia en la escena observada, podemos determinar el movimiento de un objeto. Una escena podría ser una o imagen visual (obtenida con una cámara) o cualquier otro parámetro fisico susceptible de ser medido, tal como las características electromagnéticas que existen cuando un objeto se encuentra en una posición y orientación determinadas.

Debido a que se necesitan imágenes tomadas por cámaras y un complejo algoritmo de identificación de cambios, descartamos esta técnica.

2.3.2.4 Proximidad

La técnica de proximidad [13] implica la determinación de cuando un objeto está “cerca” de una localización conocida. La presencia del objeto se aprecia utilizando un fenómeno físico con una cobertura limitada. Existen tres métodos generales para determinar la proximidad entre puntos:

1. Detectando el contacto físico. Es el método más básico de proximidad y se suelen utilizar sensores de presión, sensores de tacto y detectores de capacitancia.
2. Monitorizando puntos de acceso celulares inalámbricos. La monitorización de un dispositivo móvil, para determinar si está en el rango de uno o más puntos de acceso en una red celular inalámbrica, es otra implementación de la técnica de localización de proximidad.
3. Observando sistemas automáticos de identificación. Algunos ejemplos de este sistema son los terminales de puntos de venta con tarjeta de crédito, los historiales de login en computadores y los registros de llamadas de teléfono.

Dependiendo de los elementos que podamos instalar en el espacio que nos interese localizar, algunas de estas técnicas pueden ayudarnos a perfeccionar las estimaciones realizadas mediante otros procedimientos.

En nuestro caso, como no poseemos ninguno de los elementos anteriormente mencionados, descartamos la técnica de proximidad.

3 DISEÑO

En este capítulo se va a indagar en el diseño del sistema desarrollado en este proyecto, describiendo su estructura y la conexión entre los distintos elementos. En primer lugar, se introducirá el lenguaje de programación utilizado en el contexto de este proyecto, presentando las necesidades y limitaciones asociadas al mismo. Finalmente, se elegirán los métodos de posicionamiento que se desarrollarán en este proyecto.

3.1 Python

Aunque ya se adelantó en el apartado 1.3, el lenguaje elegido para la implementación es Python. Es administrado por la Python Software Foundation y posee una licencia de código abierto, denominada Python Software Foundation License, favoreciendo así el desarrollo de miles de módulos de terceros.

Se trata de un lenguaje de programación que destaca por la sencillez de su sintaxis y además es:

- multiparadigma, que es aquel lenguaje que permite crear programas usando más de un estilo de programación.
- interpretado, es decir, capaz de analizar y ejecutar otros programas sólo realizando la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción. Usando un intérprete, un solo archivo fuente puede producir resultados iguales incluso en sistemas sumamente diferentes. Usando un compilador, un solo archivo fuente puede producir resultados iguales solo si es compilado a distintos ejecutables específicos a cada sistema.
- de tipado dinámico en el que una misma variable puede tomar valores de distinto tipo en distintos momentos de ejecución.
- y multiplataforma. Por ejemplo, una aplicación multiplataforma puede ejecutarse sin problema alguno, tanto en Microsoft Windows en la arquitectura x86, como en Linux en la arquitectura x86 y Mac OS X ya sea en el PowerPC o sistemas Apple Macintosh basados en x86. En general, una aplicación multiplataforma se puede ejecutar tanto en todas las plataformas existentes, como en tan solo o al menos dos plataformas.

Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son códigos, la mayoría escritos también en Python, que proveen de ciertas funciones y clases para realizar determinadas tareas. Con ello, otro objetivo del diseño del lenguaje es la facilidad de extensión, pudiéndose escribir nuevos módulos fácilmente en C o C++. Un ejemplo es el módulo Tkinter27 [14], que permite crear interfaces gráficas basadas en la biblioteca Tk.

Otro aspecto importante a destacar de este lenguaje de programación, es la posibilidad de enfocarlo como lenguaje orientado a objetos (multiparadigma). Permite crear clases, añadirle métodos, heredar, instanciar objetos de esas clases… tal y como si se tratara de otro lenguaje de programación orientado a objetos como lo es Java.

3.1.1 Desventajas

Para poseer las grandes ventajas que tiene este lenguaje de programación, demos prescindir de otras o simplemente aceptar algunas limitaciones. Éstas son:

- Velocidad de ejecución [15]. Se comentó en el apartado anterior que se trata de un lenguaje interpretado, por lo que su rendimiento no puede competir con lenguajes más tradicionales que son estáticos y compilados ya que requiere de un programa especial llamado intérprete para traducir en tiempo real las instrucciones, que nunca son compiladas. Este programa analiza el código Python y prevé cierto tipo de información que no es provista por el programador, como el tipo de datos o el tamaño de los arrays y pasa las instrucciones al procesador. Todo este trabajo extra resulta en una menor velocidad de ejecución.
- Otro inconveniente de Python en comparación con otros lenguajes es lo débil que es para hacer programas donde sea necesario ejecutar múltiples hilos.
- Para muchos, la documentación oficial existente no es tan completa como la de otros lenguajes de programación, como Java o PHP.

3.1.2 Alternativas

Existen muchos lenguajes de programación e igual número de alternativas. Los principales objetivos por los cuales Python ha sido elegido frente a cualquier otro, son:

- su simple e intuitiva sintaxis frente a C o Java, teniendo que prestar atención casi únicamente al indentado.
- el tipado dinámico, ya explicado en el apartado 3.1, permite asignar varios tipos a una misma variable, o simplemente obviar el tipo de la variable, ya que el intérprete le asignará el que le corresponda.
- Innumerable cantidad de módulos externos, permitiéndonos abarcar casi cualquier temática a la hora de desarrollar una aplicación.
- El soporte actual es muy efectivo debido al amplio uso global de este lenguaje y constante actualizaciones.

3.1.3 Módulos

Llegados a este punto se van a detallar las librerías o módulos externos empleados en la elaboración de este proyecto.

- datetime: obtener hora y fecha del equipo.
- pyplot de matplotlib: dibuja gráficas y permite su configuración.
- numpy: realiza operaciones complejas y con matrices.
- math: también realiza otros tipos de operaciones como raíces, potencias y valor absoluto.
- json y enconder: usados para exportar y dar formato a la variable que contendrá los resultados.

A continuación, se detallarán los módulos creados para lograr el objetivo del proyecto:

- main: muestra el menú y llama a otros módulos para completar la acción
- propagation: realiza una simulación y devuelve los resultados.

- trilateration: es un módulo que realiza la trilateración a partir de varias circunferencias para todos los métodos de localización.
- locationHandler: módulo que hace de intermediario entre main y trilateration. Calcula todos los parámetros previos necesarios para realizar la posterior estimación de la posición.
- …

Cada uno de los módulos detallados anteriormente, serán explicados con mayor profundidad en el apartado de Implementación.

3.1.4 Decisiones

En este apartado se incluirá la elección de los métodos de localización que se implementarán y se describirán en el apartado 4. Teniendo en cuenta la Tabla 2 del apartado 2.2.9, podemos obtener ciertas conclusiones que nos ayudarán a descartar o no, algunos de los métodos descritos anteriormente.

Por ejemplo, debido a que el precio de los APs MIMO son mucho más elevados que los convencionales, descartamos **CSI** y **AoA** como métodos a aplicar en este proyecto. Además, también necesitan acceder a parámetros de muy bajo nivel que son difíciles de obtener en un AP convencional.

Por consiguiente, por razones de necesitar marcas de tiempo muy precisas, además de su complejidad, **TDoA** tampoco será objeto de estudio en este proyecto.

También se descarta el método de posicionamiento GPS ya que en interiores la señal recibida por los satélites no es la adecuada y es propenso a muchos errores.

Por tanto, los métodos que se explicarán, aplicarán y desarrollarán en este proyecto son: **RSSI (Calibración)**, **ToF** y **Pesos**.

El **Fingerprinting** también es descartado debido a que necesitamos medir todo el mapa de localización y no es eficiente si éste cambia con frecuencia.

4 IMPLEMENTACIÓN

La finalidad de este capítulo es la de detallar todos los aspectos de la implementación del sistema, siguiendo los elementos indicados en el capítulo 3. Se hará hincapié, tanto en la organización de las distintas partes del código, como en su funcionalidad. La organización de este apartado se rige por el orden seguido durante el desarrollo del proyecto, comenzando con la parte principal del programa Python, continuando con los algoritmos de localización y posicionamiento, y terminando con simulaciones más completas y extensas.

4.1 Aplicación Python

4.1.1 Estructura de los archivos

Para comenzar este apartado, se ofrece una primera visión de la estructura de archivos de la aplicación. Los archivos Python de la aplicación están estructurados de la siguiente forma:

Figura 25. Estructura del proyecto

4.2 Módulos

A continuación, se detallarán todos los módulos creados para la correcta ejecución de esta aplicación.

4.2.1 main.py

Módulo principal que muestra un menú con todas las acciones que la aplicación ofrece.

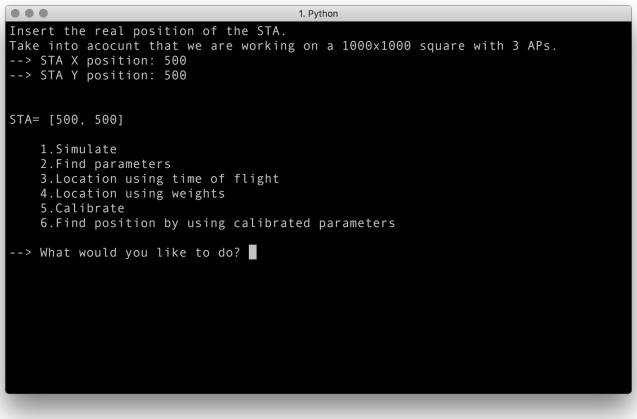


Figura 26. Menú principal.

Las acciones que se pueden llevar a cabo son:

- En cuanto se ejecuta el programa, se solicita la posición original del STA, que será la estimada por los algoritmos usados en este proyecto, con el fin de calcular porcentajes de error y efectividad de los mismos. Además, se asignan valores a una serie de parámetros como son la frecuencia, potencia de transmisión, posición de los APs y margen de error. Estos valores se mantendrán fijos para todas las opciones que se muestran en el menú salvo en los casos que se indique lo contrario. Todos los métodos aplicados se realizan sobre un mapa de 1000x1000 con 3 APs repartidos en las 3 esquinas inferiores. Las unidades de distancia siempre serán consideradas en centímetros. Para todos los métodos de localización, el método de posicionamiento elegido es, como se ha indicado anteriormente, el algoritmo de trilateración.
- **Opción 1 - Simulate**: consiste en realizar una simulación de manera que nos devuelva los parámetros necesarios para aplicar los algoritmos de localización y posteriormente el de posicionamiento o trilateración. Estos parámetros son básicamente el tiempo de vuelo (ToF) o el RSSI. El módulo *propagation.py* es el encargado de realizar estas simulaciones.
- **Opción 2 - Find parameters** (distancia según RSSI): Se encarga de llamar a la función (*findParams* del módulo *locationHanbdler.py*) que calcula la distancia a partir del RSSI que ha sido simulado previamente mediante la ecuación [2]. Finalmente muestra los posibles candidatos de ser el punto correctamente estimado.
- **Opción 3 - Location using time of flight**: Prepara los parámetros para realizar un barrido de todo el mapa de posicionamiento. Esto se lleva a cabo haciendo una simulación usando el módulo propagation.py para cada posición posible del STA. Dada una posición, se obtienen los tiempos de vuelo entre STA y APs, para luego estimar la distancia usando la ecuación [3] y calcular la posición mediante el algoritmo de trilateración, a través del módulo *locationHandler.py* (función *distanceUsingTimeOfFlight*). Se compara la posición estimada con la original, y si está dentro de un rango que consideremos aceptable (en torno a 1m), añadimos un identificador en el fichero para que sea fácil detectar cuando el algoritmo ha devuelto una posición aceptable. Esto se repite para cada

posición del plano. Una gráfica con el porcentaje de estimaciones correctas se mostrará al terminar todo el proceso.

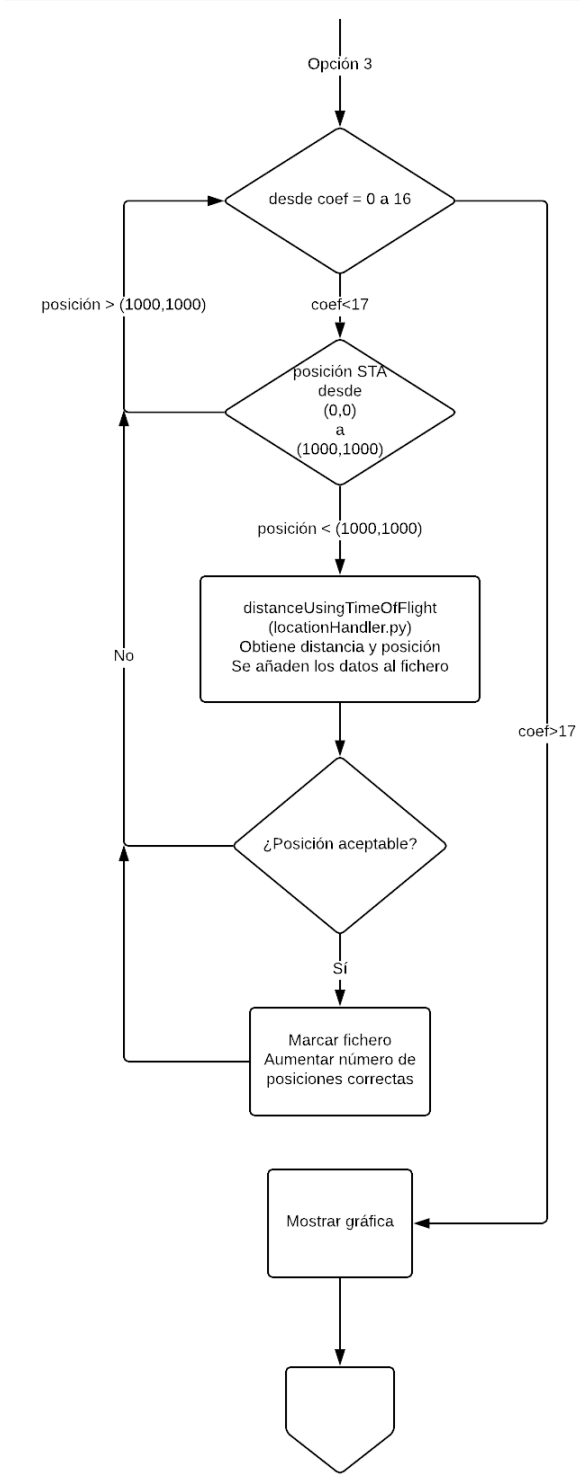


Figura 27. Proceso seguido en el método tiempo de vuelo (opción 3).

- **Opción 4 - Location using weights**: Se encarga de llamar a la función que calcula la posición del STA a partir de los pesos asignados (*positionUsingWeights* del módulo *locationHanbdler.py*) según la potencia recibida (RSSI).
- **Opciones 5 y 6 - Calibrate y Find position by using calibrated parameters**: usadas en ese orden para hacer la calibración (función *calibrate* del módulo *locationHanbdler.py*) y seguidamente obtener los resultados de un barrido completo del mapa de posicionamiento usando el método de localización basado en RSSI. Seguidamente se compara los resultados y se añaden en un fichero de la misma manera que se hace en la opción anterior. Finalmente se muestran las estadísticas resultantes. Se puede ver el proceso de calibración (opción 5) en la Figura 20.

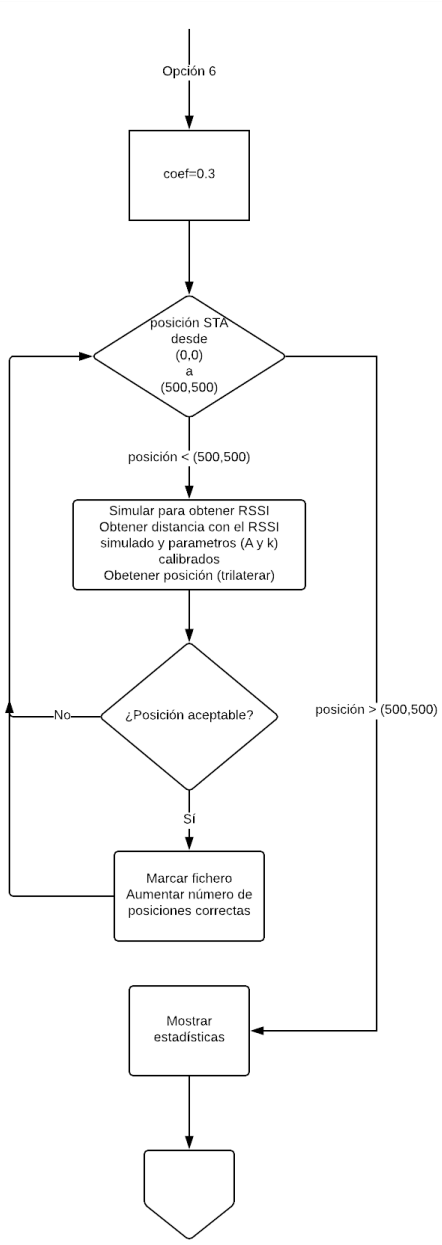


Figura 28. Proceso seguido en la opción 6.

4.2.2 locationHandler.py

Este módulo se ha implementado básicamente para hacer lo más modular posible la aplicación, es decir, repartir la carga de código en varios módulos, de manera que esté divido según las diferentes funcionalidades de los mismos.

Por tanto, en este apartado una serie de funciones ya nombradas, que son usadas por el módulo anterior para completar el objetivo final. Estas funciones son:

- findParams: esta función es llamada por la opción 2 y se encarga de crear el archivo donde se volcarán los resultados (*results/findingResults_+FechaAcual.txt*), hace un barrido de los parámetros A y n de la ecuación [2] y calcula la distancia para cada iteración. Finalmente obtiene la posición, es decir, realiza la trilateración, mediante la función *tri4Simulate* del módulo *trilateration.py*.

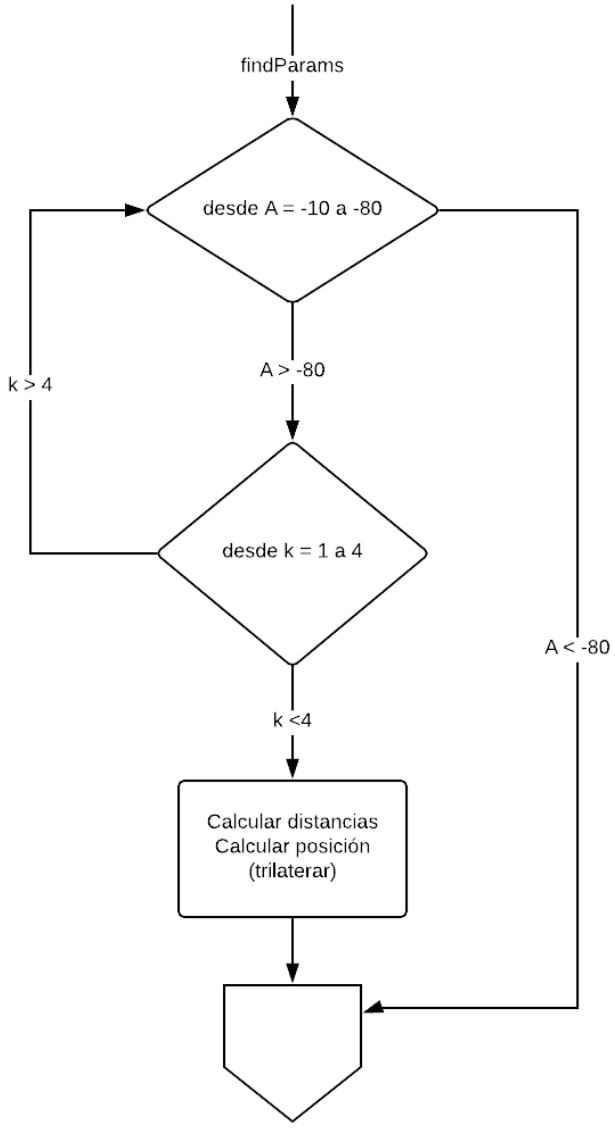


Figura 29. Función findParams.

- distanceUsingTimeOfFlight: función usada en la opción 3 para calcular la distancia según el tiempo de propagación devuelvo por la simulación. También se encarga de llamar a la función encargada de trilaterar para el caso de localización usando el tiempo de vuelo (*tri4TOF* del módulo *trilateration.py*)
- positionUsingWeights: se trata de la función que prepara las ecuaciones de las cuales van a resultar los pesos que se le asignará a cada AP para determinar la posición estimada del STA.
- computePOW: función auxiliar usada por la anterior para automatizar las potencias de ecuaciones en las que varía el exponente.
- computeEquation: similar a la anterior, pero en este caso, no sólo varía el exponente, sino la base también.

Estas dos últimas, también obtiene los pesos que se le asigna a cada AP y muestra la posición estimada resultante del STA.

- calibrate: función intermedia para llevar a cabo la calibración. Primero, se encarga de realizar un barrido de los parámetros al igual que se lleva a cabo en la función *findParams* (Figura 28). Después, para cada uno de estos valores, se calcula su posición estimada y se valida mediante la función *validate* del módulo *trilateration.py*. Al final de las iteraciones, se devuelven una lista de las posiciones aceptadas y los parámetros para las que lo han sido.

4.2.3 trilateration.py

Este módulo consta de dos partes:

- En primer lugar, una serie de clases y funciones cuyo objetivo es calcular el punto de intersección de varias circunferencias o el centro del polígono que forman las mismas si éstas no cortan en un solo punto. Es lo que se conoce como trilateración (apartado 2.3.1).
- En segundo y último lugar, se encuentras las funciones que utilizan las anteriores para obtener el objetivo, que es el punto resultante de la trilateración. Éstas pueden realizar otras acciones como alguna comprobación adicional, dibujar gráficas o escribir en ficheros.

4.2.3.1 Clases y funciones para definir la trilateración

A continuación, se van a detallar brevemente las clases y funciones empleadas para conseguir realizar la trilateración. Él código usado para llevar esta compleja acción a cabo, es una versión modificada de la propuesta por Noomrevlis [12].

- La clase **base_station** corresponde con un AP. Contiene sus coordenadas y la distancia estimada asociada.
- La clase **point** tiene como objetivo almacenar las coordenadas (x,y) de un punto cualquiera.
- La clase **circle** es usada para instanciar circunferencias. Contiene un punto (clase point) y un radio.
- La clase **json_data** está destinada para volcar todos los resultados a una variable en formato json. Contiene la lista de todas las circunferencias (APs), todos los puntos de cortes entre ellas, y el centro, que corresponde con la posición del STA.
- La función **serialize_instance** permite la serialización de JSON, es decir, permite guardar el contenido de objetos JSON en variables con ese mismo formato.

- **get_two_points_distance**: devuelve la distancia entre dos puntos.

  Sean dos puntos con coordenadas (x_1,y_1) y (x_2,y_2), la distancia entre ellos se puede calcular con la siguiente ecuación:

  $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad [4]$$

- **get_two_circles_intersecting_points**: obtiene los puntos de intersección de dos circunferencias. Se detalla a continuación el proceso para conseguir los dos puntos de intersección de dos circunferencias.

  Primero, suponemos que tenemos dos circunferencias, una situada en (x_1,y_1) y la otra en (x_2,y_2) cuyos radios son r_1 y r_2 respectivamente. Sus ecuaciones serían las siguientes:

  $$x_1^2 + y_1^2 = r_1^2 \quad [5]$$

  $$x_2^2 + y_2^2 = r_2^2 \quad [6]$$

  Teniendo en cuenta la ecuación [4], definimos a continuación:

  $$a = \frac{r_1^2 - r_2^2 + d^2}{2d} \quad [7]$$

  $$h = \sqrt{r_1^2 + a^2} \quad [8]$$

  Ahora tenemos dos vectores unitarios ortogonales con los que podemos trasladar y rotar para obtener la solución. Estos vectores son:

  $$\left(\frac{x_2 - x_1}{d}\right), \left(\frac{y_2 - y_1}{d}\right) \text{ y } \left(\frac{y_2 - y_1}{d}\right), -\left(\frac{x_2 - x_1}{d}\right)$$

  La solución buscada es la siguiente:

  $$x = \frac{a}{d}(x_2 - x_1) \pm \frac{h}{d}(y_2 - y_1) + x_1 \quad [9]$$

  $$y = \frac{a}{d}(y_2 - y_1) \mp \frac{h}{d}(x_2 - x_1) + y_1 \quad [10]$$

- **get_all_intersecting_points**: devuelve todos los puntos de intersección existentes entre las circunferencias. Se puede lograr aplicando los procedimientos detallados en las dos funciones anteriores para cada par de circunferencias existentes.
- **is_contained_in_circles**: devuelve si un punto está contenido en todas y cada una de las circunferencias de una lista, comprobando si la distancia desde punto al centro de la circunferencia es mayor que el radio de la misma más un pequeño margen de error.
- **get_polygon_center**: obtiene el centro del polígono resultante de todos los puntos de intersección. El resultado es la media de todos los puntos.

4.2.3.2 Funciones para realizar la trilateración

Llegado a este punto, se va a explicar el funcionamiento de las funciones empleadas para obtener la posición del STA, independientemente del método de localización elegido para calcular las distancias a cada uno de los APs.

El procedimiento de todas estas funciones es similar, y se puede resumir de la siguiente manera:

1. Crear lista de coordenadas X e Y (xList, yList) de cada AP. También una lista con todas las distancias (R), que será el radio de las circunferencias.
2. Crear lista de puntos (P) para cada par de valores en xList e yList. Se crea también una lista de circunferencias (C) para cada punto y su radio (R).
3. Obtener todos los puntos de intersección.
4. Comprobar si esos puntos están contenidos en las circunferencias. Si lo están, se añade a una lista de puntos interiores (inner_points)
5. Obtener el centro del polígono que forman los puntos interiores y volcar los resultados en una variable JSON.
6. Extraer del fichero el punto deseado eliminando partes del fichero hasta encontrar el punto.
7. Devolver el punto estimado.

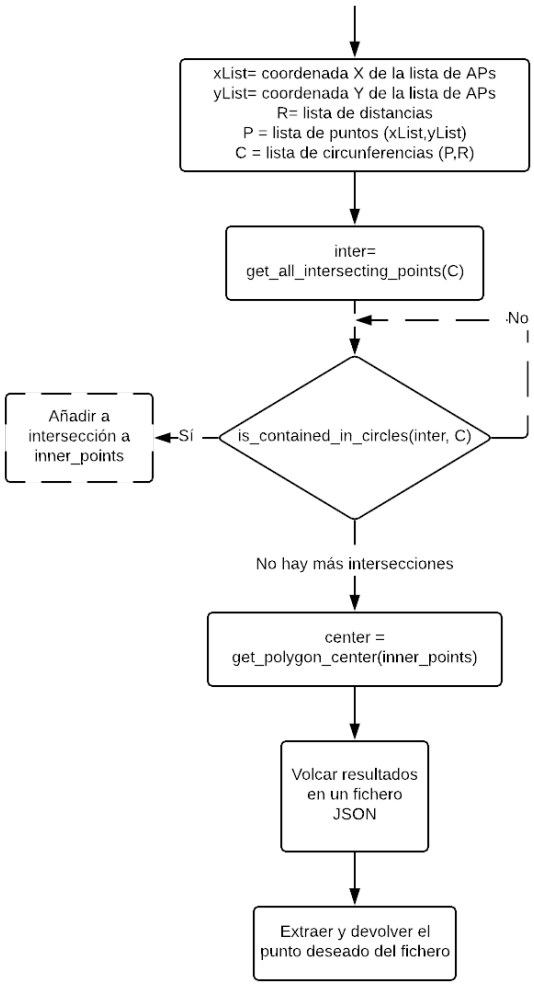


Figura 30. Procedimiento de triangulación.

Por tanto, todas las funciones que se van a comentar a continuación, realizan el proceso previo que acabamos de comentar y además otras funcionalidades como:

- **tri4Simulate**. Es la usada en la opción de localización mediante RSSI.
- **tri4TOF**: posee una función opcional de dibujar en un grafo el resultado de la trilateración. Es usada en la opción 3 (localización usando el tiempo de vuelo), además de en las simulaciones
- **validate**: le permite a la opción 5 (calibrar) obtener si los resultados de trilateración han sido válidos o no.

4.2.4 propagation.py

Este módulo es usado para realizar una simulación del modelo de propagación en un canal 802.11.

Para un canal de 2.4 GHz, las ecuaciones que modelan la pérdida de la trayectoria (PathLoss) son [16]:

$$PL(d) = 40.05 + 20\log\left(\frac{fc}{2.4}\right) + 20\log(min(d,5)) + (d > 5) + 35\log\left(\frac{d}{5}\right) \quad [11]$$

donde:

- fc es la frecuencia con la que emite el STA en GHz.
- d es la distancia del STA al AP en m.

Para estimar el retardo de propagación:

$$retardo = 2(1 + coef * random)\frac{d}{c} \quad [12]$$

donde:

- coef es un coeficiente que, junto con la variable random, permiten que el retardo no sea siempre el ideal.
- random es una variable aleatoria entre 0 y 1 que permite otorgarle mayor o menos peso al coeficiente de variación del retardo.
- c es la velocidad de la luz en m/s.

Y para el caso del RSSI:

$$RSSI\ (dBm) = Ptx - L_{space} - L_{fading} * N(0,1) \quad [13]$$

donde:

- Ptx es la potencia de transmisión de los Aps
- $L_{space} = PL(d) + L_{walls}$, con L_{walls} siendo pérdidas por paredes o muros.
- L_{fading} son las pérdidas por desvanecimiento.
- $N(0,1)$ es una distribución normal con media 0 y varianza 1.

4.3 Vista general

En este apartado se desea añadir un esquema de manera que todo lo expuesto en los apartados anteriores sea resumido, pudiendo así tener una visión más clara de la estructura de la aplicación.

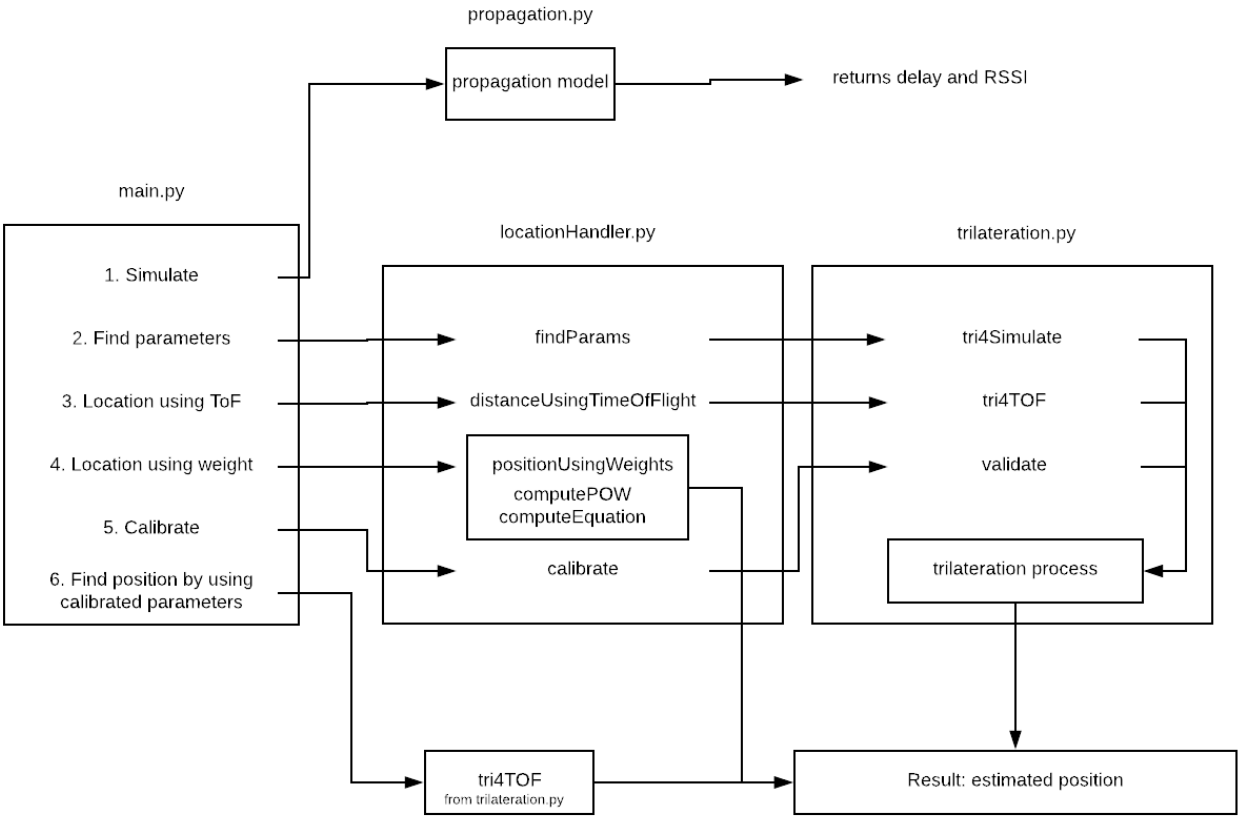


Figura 31. Esquema resumen.

5 SIMULACIONES

En este capítulo, se proponen un conjunto de simulaciones más complejas que permiten certificar el correcto funcionamiento del sistema, comprobando que los resultados son los esperados y está preparado para funcionar en casos de uso reales. Las simulaciones están enfocadas tanto a componentes individuales, como al sistema completo, de manera que podamos conocer el rendimiento y las limitaciones de éstos.

5.1 Simulación 1: Tiempo de vuelo variando el número de puntos de accesos

Esta simulación estará basada en método de localización según el tiempo de vuelo. En ella variamos el número de APs para cada simulación para determinar cómo de preciso es este método según vamos cambiando el número de APs y su posición.

El desarrollo es el mismo que se ha definido en el apartado 4.2.1, con la diferencia que el número de APs va cambiando de una simulación a otra. La posición de los mismos se encuentra definida en un fichero de texto (coordinates.txt) y el programa es el encargado de obtenerlas mediante el siguiente conjunto de funciones localizadas en el módulo locationHandler.py. Estas funciones son exclusivas de esta simulación, y no aparecen en el módulo que es utilizado en el programa general.

Antes de ejecutar ninguna función, es necesario obtener una lista con todas las líneas del fichero anterior.

Además, hay que tener en cuenta el formato en el que es escrito el fichero de texto, para poder obtener los datos posteriormente. Cada línea de texto se define de la siguiente manera:

X Y - X Y - X Y

donde X e Y son las coordenadas para cada uno de los APs y el delimitador que permite distinguir los diferentes APs es un guion "-".

El coeficiente del módulo propagación es constante e igual a 0.3

Las funciones que permiten obtener las posiciones del fichero de texto son:

- splitString: Trocea la lista de posiciones usando como delimitador el carácter de espacio.
- delete_: Detecta si en la cadena que se le pasa por parámetro existe el carácter de guion y lo elimina
- buildString: Construye la lista final con las posiciones de los APs según el número de la iteración en la que se encuentre el programa principal.
- parser: función principal que utiliza las anteriores en el orden adecuado para garantizar la correcta extracción de los datos del fichero de texto.

Por último, la simulación guarda el resultado de todas las simulaciones, de manera que muestra una gráfica con el número de estimaciones correctas para las diferentes iteraciones. Es decir, una gráfica con el porcentaje de estimaciones válidas en función del número de APs.

```
-----------------------------------------------
-----------Starting SIMULATION 1--------------
-----------------------------------------------

Loaded 10 iterations...
Press Enter to start...

APs location ----> [[0, 0], [500, 500]]
For 2 APs ----> 10000 iterations and 1356 correct ones (13.56 %)

APs location ----> [[0, 0], [0, 500], [1000, 0]]
For 3 APs ----> 10000 iterations and 3234 correct ones (32.34 %)

APs location ----> [[0, 0], [0, 1000], [1000, 0], [1000, 1000]]
For 4 APs ----> 10000 iterations and 8004 correct ones (80.04 %)

APs location ----> [[0, 0], [0, 1000], [1000, 0], [1000, 1000], [500, 1000]]
For 5 APs ----> 10000 iterations and 8405 correct ones (84.05 %)

APs location ----> [[0, 0], [0, 1000], [1000, 0], [1000, 1000], [500, 1000], [500, 0]]
For 6 APs ----> 10000 iterations and 9009 correct ones (90.09 %)

APs location ----> [[0, 0], [0, 1000], [1000, 0], [1000, 1000], [500, 1000], [500, 0], [500, 500]]
For 7 APs ----> 10000 iterations and 9468 correct ones (94.68 %)

APs location ----> [[0, 0], [0, 1000], [1000, 0], [1000, 1000], [500, 1000], [500, 0], [500, 500], [0, 500]]
For 8 APs ----> 10000 iterations and 9639 correct ones (96.39 %)

APs location ----> [[0, 0], [0, 1000], [1000, 0], [1000, 1000], [500, 1000], [500, 0], [500, 500], [0, 500], [1000, 500]]
For 9 APs ----> 10000 iterations and 9747 correct ones (97.47 %)
```

Figura 32. Ejecución de la simulación 1: línea de comandos.

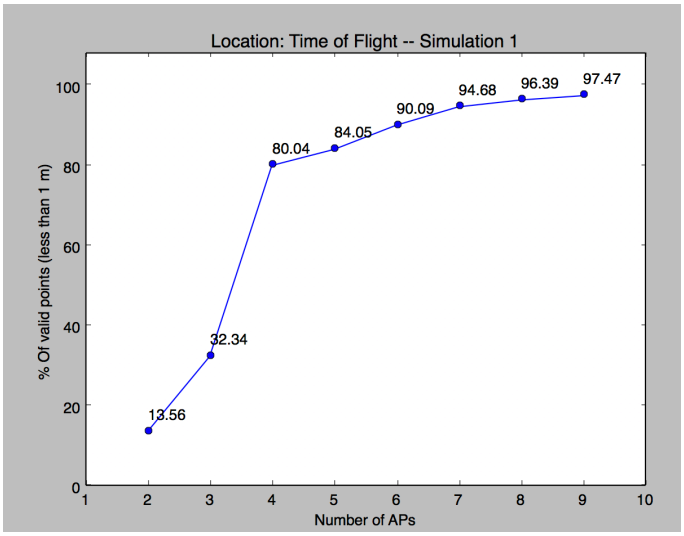


Figura 33. Ejecución de la simulación 1: gráfica resultante

Como se puede observar, conforme aumentamos el número de puntos de acceso el porcentaje de acierto va aumentando. Esto se debe a que cuantos más APs, el área de intersección resultante es menor, y poseemos más información desde otros puntos del mapa, siendo más fácil determinar su posición.

Todo ello es posible teniendo en cuenta que según incrementamos el número de APs, el tiempo de ejecución del algoritmo aumenta notablemente.

5.2 Simulación 2: Variación de la aleatoriedad del tiempo de vuelo

Teniendo en cuenta la descripción de la simulación anterior, a la segunda se le añade las siguientes modificaciones:

- Para un mismo número de APs, se hace un conjunto de simulaciones variando el coeficiente de aleatoriedad del tiempo de vuelo en el módulo *propagation.py*. El objetivo de esta simulación es, por tanto, ver cómo afecta la variación de éste a la hora de obtener la posición de un STA.
- En este caso, el programa guarda una gráfica según vamos aumentando el número de puntos de accesos, y no una sola gráfica con toda la variación como en la simulación anterior. Es decir, en cada una de ellas podemos ver como varía el porcentaje de posiciones válidas según la variación del coeficiente para un mismo número de APs.

```
-----------------------------------------------
-----------Starting SIMULATION 2---------------
-----------------------------------------------
All previous PNG files have veen deleted

Loaded 10 iterations...
Press Enter to start...
For 2 APs, coef 0.1  ----> 1600 iterations and 320 correct ones (20.0 %)
For 2 APs, coef 0.2  ----> 1600 iterations and 250 correct ones (15.625 %)
For 2 APs, coef 0.3  ----> 1600 iterations and 216 correct ones (13.5 %)
For 2 APs, coef 0.4  ----> 1600 iterations and 186 correct ones (11.625 %)
For 2 APs, coef 0.5  ----> 1600 iterations and 148 correct ones (9.25 %)
For 2 APs, coef 0.6  ----> 1600 iterations and 155 correct ones (9.6875 %)
For 2 APs, coef 0.7  ----> 1600 iterations and 127 correct ones (7.9375 %)
For 2 APs, coef 0.8  ----> 1600 iterations and 118 correct ones (7.375 %)
For 2 APs, coef 0.9  ----> 1600 iterations and 124 correct ones (7.75 %)
For 2 APs, coef 1.0  ----> 1600 iterations and 100 correct ones (6.25 %)
For 3 APs, coef 0.1  ----> 1600 iterations and 720 correct ones (45.0 %)
For 3 APs, coef 0.2  ----> 1600 iterations and 621 correct ones (38.8125 %)
For 3 APs, coef 0.3  ----> 1600 iterations and 553 correct ones (34.5625 %)
For 3 APs, coef 0.4  ----> 1600 iterations and 444 correct ones (27.75 %)
For 3 APs, coef 0.5  ----> 1600 iterations and 384 correct ones (24.0 %)
For 3 APs, coef 0.6  ----> 1600 iterations and 348 correct ones (21.75 %)
For 3 APs, coef 0.7  ----> 1600 iterations and 280 correct ones (17.5 %)
For 3 APs, coef 0.8  ----> 1600 iterations and 248 correct ones (15.5 %)
For 3 APs, coef 0.9  ----> 1600 iterations and 218 correct ones (13.625 %)
For 3 APs, coef 1.0  ----> 1600 iterations and 187 correct ones (11.6875 %)
For 4 APs, coef 0.1  ----> 1600 iterations and 1581 correct ones (98.8125 %)
For 4 APs, coef 0.2  ----> 1600 iterations and 1525 correct ones (95.3125 %)
For 4 APs, coef 0.3  ----> 1600 iterations and 1280 correct ones (80.0 %)
For 4 APs, coef 0.4  ----> 1600 iterations and 1082 correct ones (67.625 %)
For 4 APs, coef 0.5  ----> 1600 iterations and 858 correct ones (53.625 %)
For 4 APs, coef 0.6  ----> 1600 iterations and 690 correct ones (43.125 %)
For 4 APs, coef 0.7  ----> 1600 iterations and 555 correct ones (34.6875 %)
For 4 APs, coef 0.8  ----> 1600 iterations and 501 correct ones (31.3125 %)
For 4 APs, coef 0.9  ----> 1600 iterations and 406 correct ones (25.375 %)
For 4 APs, coef 1.0  ----> 1600 iterations and 353 correct ones (22.0625 %)
For 5 APs, coef 0.1  ----> 1600 iterations and 1576 correct ones (98.5 %)
For 5 APs, coef 0.2  ----> 1600 iterations and 1525 correct ones (95.3125 %)
For 5 APs, coef 0.3  ----> 1600 iterations and 1378 correct ones (86.125 %)
```

Figura 34. Ejecución de la simulación 2: línea de comandos

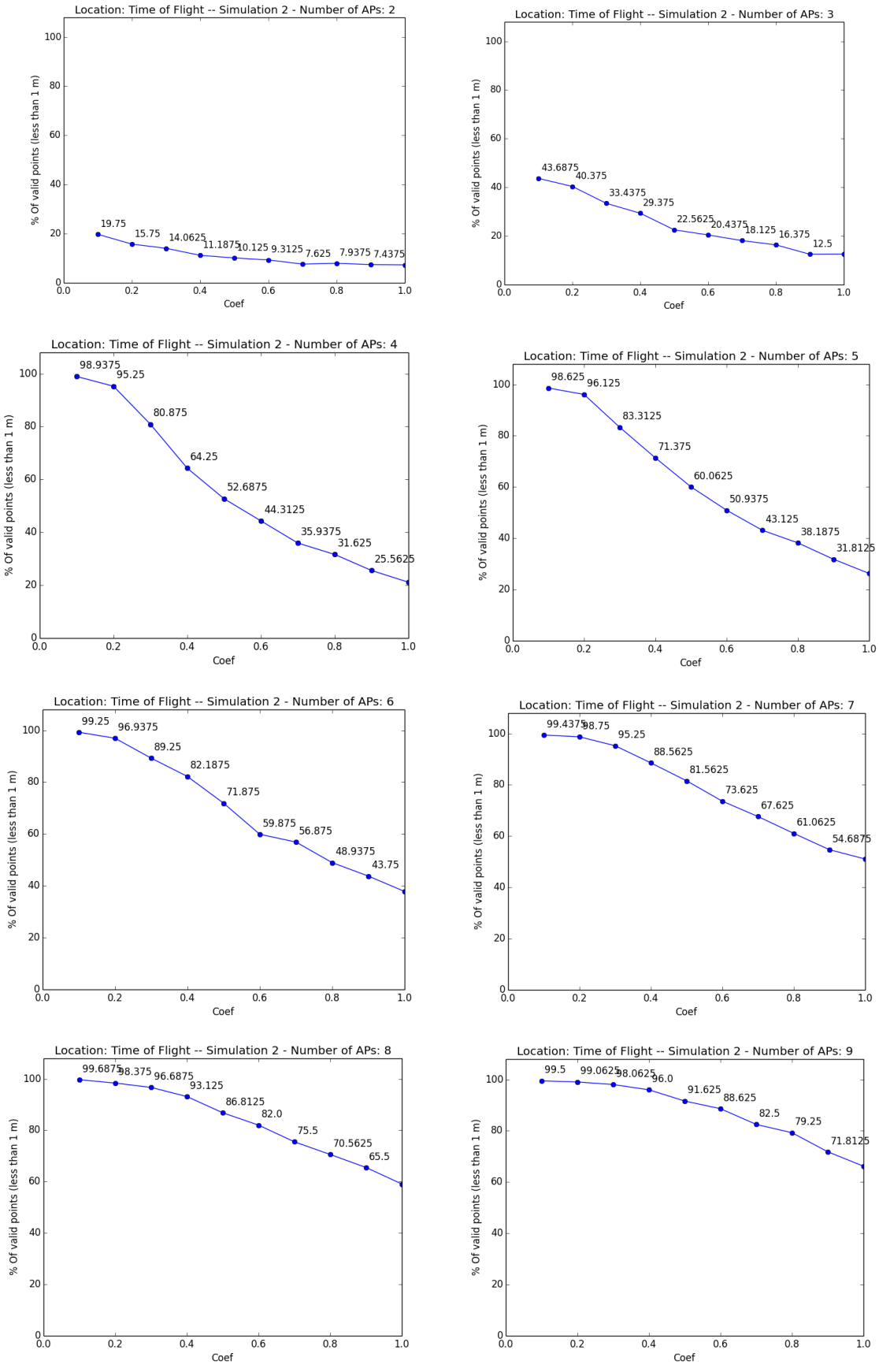


Figura 35. Ejecución de la simulación 2: gráfica para distintos números de APs.

Analizando las gráficas de los ejemplos anteriores, podemos observar que según variamos el índice que hace que el tiempo de procesado de la trama en el AP destino sea mayor o menor, en definitiva, el índice que hace que el tiempo de vuelo sea mayor o menor, el porcentaje decrece independientemente del número de APs.

5.3 Simulación 3: Variación de la posición de calibración

En el apartado 2.2.8 se proponía llevar a cabo la calibración en una posición en la que existan menos obstáculos para cada uno de los APs, o aquella en las que los diferentes APs se encuentren aproximadamente a la misma distancia del STA.

Sim embargo, en esta simulación, esto no se realizará así, sino que la posición del STA irá cambiando de un punto a otro, sin importar que exista mayor probabilidad de error en la triangulación para unos APs que para otros, o se encuentre más cerca o más lejos de los APs.

Para cada posición del STA, se estima en primer lugar los parámetros A y n de la ecuación [2]. Seguidamente, también para casa posición del STA, se calcula la distancia a cada AP. A partir de estas distancias estimadas obtenemos, mediante el algoritmo de trilateración, la posición estimada, que será comparada con la posición real del STA para comprobar si la estimación ha sido correcta o no.

El resultado se volcará en un fichero de texto con un formato similar al siguiente:

```
Real STA position (430,940)
 Iterations= 100
 Correct positions= 43
43.0%
Real STA position (530,40)
 Iterations= 100
 Correct positions= 88
88.0%
Real STA position (530,140)
 Iterations= 100
 Correct positions= 88
88.0%
Real STA position (530,240)
 Iterations= 100
 Correct positions= 90
90.0%
Real STA position (530,340)
 Iterations= 100
 Correct positions= 75
75.0%
Real STA position (530,440)
 Iterations= 100
 Correct positions= 75
75.0%
Real STA position (530,540)
 Iterations= 100
 Correct positions= 75
75.0%
Real STA position (530,640)
 Iterations= 100
 Correct positions= 75
75.0%
Real STA position (530,740)
 Iterations= 100
 Correct positions= 75
75.0%
```

Figura 36. Ejecución de la simulación 3: formato de salida.

Si se analiza el documento anterior podemos observar que existen zonas en las que el porcentaje de punto correctamente estimados decrece notablemente. Trasladadas a una gráfica, estas zonas son, aproximadamente:

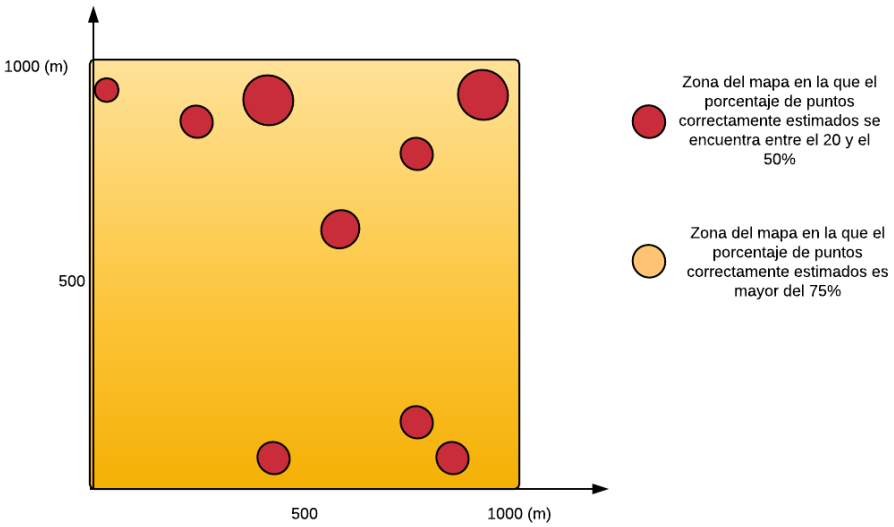


Figura 37. Ejecución de la simulación 3: mapa de porcentaje de estimaciones aceptables.

Como se puede observar, hay zonas en las que si colocamos el STA para calibrar el porcentaje de puntos correctamente estimados devueltos por el algoritmo de triangulación es bastante reducido

5.4 Simulación 4: Variación del Fading en la localización por pesos

Como se comentó en el apartado 2.2.3, según las ecuaciones que utilicemos para obtener los pesos, los resultados serán mejores o peores. Para la realización de este trabajo, se han probado numerosas ecuaciones de las cuales una arrojaba resultados aceptables. Será esta última la que se detallará y usará en la elaboración del proyecto. A continuación, una lista de las ecuaciones utilizadas, y finalmente la que se ha elegido:

Se supone que existen "*N*" APs e "*i*" y "j" son índices que hacen refencia a uno de estos APs. Por tanto: $1 \leq i \leq N$ y $1 \leq j \leq N$.

Sea W(i) la varíable que contiene el peso del AP *i*, *p* una variable usada para elevar las distintas funciones a una serie de potencias con $p = 1 \dots 10$, *X* una variable usada como base de un conjunto de potencias cuyo rango es: $X = 1 \dots 7$, y RSSI(i) el RSSI del AP *i*.

$$W(i) = (100 + RSSI(i))^p \quad [14]$$

$$W(i) = \left(\frac{1}{RSSI(i)}\right)^p \quad [15]$$

$$\mathrm{W(i)} = \log(100 + RSSI(i))^p \quad [16]$$

$$W(i) = X^{RSSI(i)} \quad [17]$$

$$W(i) = \frac{1}{\sqrt{100 + RSSI(i)}}^{p} \quad [18]$$

$$W(i) = \log\left(\sqrt{100 + RSSI(i)}\right) \quad [19]$$

$$W(i) = \frac{1}{\log\left(\sqrt{100 + RSSI(i)}\right)} \quad [20]$$

Estas ecuaciones anteriores, o no nos permiten calcular los pesos correctamente o bien no siguen un patrón fácil de encontrar. Es por esta razón por las que son descartadas. Los resultados pueden ser comprobados ejecutando el programa principal.

Sin embargo, hay una ecuación que sí nos arroja buenos resultados. Es la utilizada en esta quinta simulación, una de las que se maneja en la empresa Galgus[11] como candidata para su producto de localización, la cual es complementada por varias gráficas para ver de una manera más clara el resultado obtenido. Las ecuaciones son:

$$a(i) = 10^{-\frac{RSSI(i)}{20}} \quad [21]$$

$$b(i) = \frac{\sum_{j=1}^{N} a(j)}{a(i)} \quad [22]$$

$$W(i) = \frac{b(i)}{\sum_{j=1}^{N} b(j)} \quad [23]$$

Una vez obtenidos los pesos, nos queda, simplemente, obtener la posición estimada del STA (x e y). Esto se logra mediante las siguientes ecuaciones, teniendo en cuenta que x_i e y_i son las coordenadas del AP i, respecto del eje X e Y.

$$x = \sum_{i=1}^{N} x_i * W(i) \quad [24]$$

$$y = \sum_{i=1}^{N} y_i * W(i) \quad [25]$$

Aplicando estas fórmulas, y teniendo en cuenta el término L_{fading}[12] de la ecuación [13], esta simulación consiste en variar el número de APs en el mapa de localización, y para un mismo número de APs, cambiar el L_{fading} del módulo *propagation.py* con el objetivo de generar las medidas del RSSI simulado y concluir cómo afecta éste al porcentaje de puntos estimados correctamente. Es decir, para cada valor de Lfading se lleva a cabo la estimación de posición para cada una de las posiciones posible de STA, y llegar a un valor de porcentaje de aciertos para cada uno de esos valores. Aunque el valor recomendado por la IEE para este parámetro es 5, se hace un barrido de su valor para comprobar cómo afecta éste a las estimaciones.

Los resultados se guardan en un fichero de texto dentro del directorio *results*, además de una serie de gráficas para contrastar los resultados obtenidos.

[11] Galgus: empresa sevillana que ha desarrollado un software que permite tener una conexión wifi más rápida, eficiente y de menor consumo eléctrico.

[12] Lfading: variable que aplica cierta cantidad de ruido o desvanecimiento en el cálculo del RSSI.

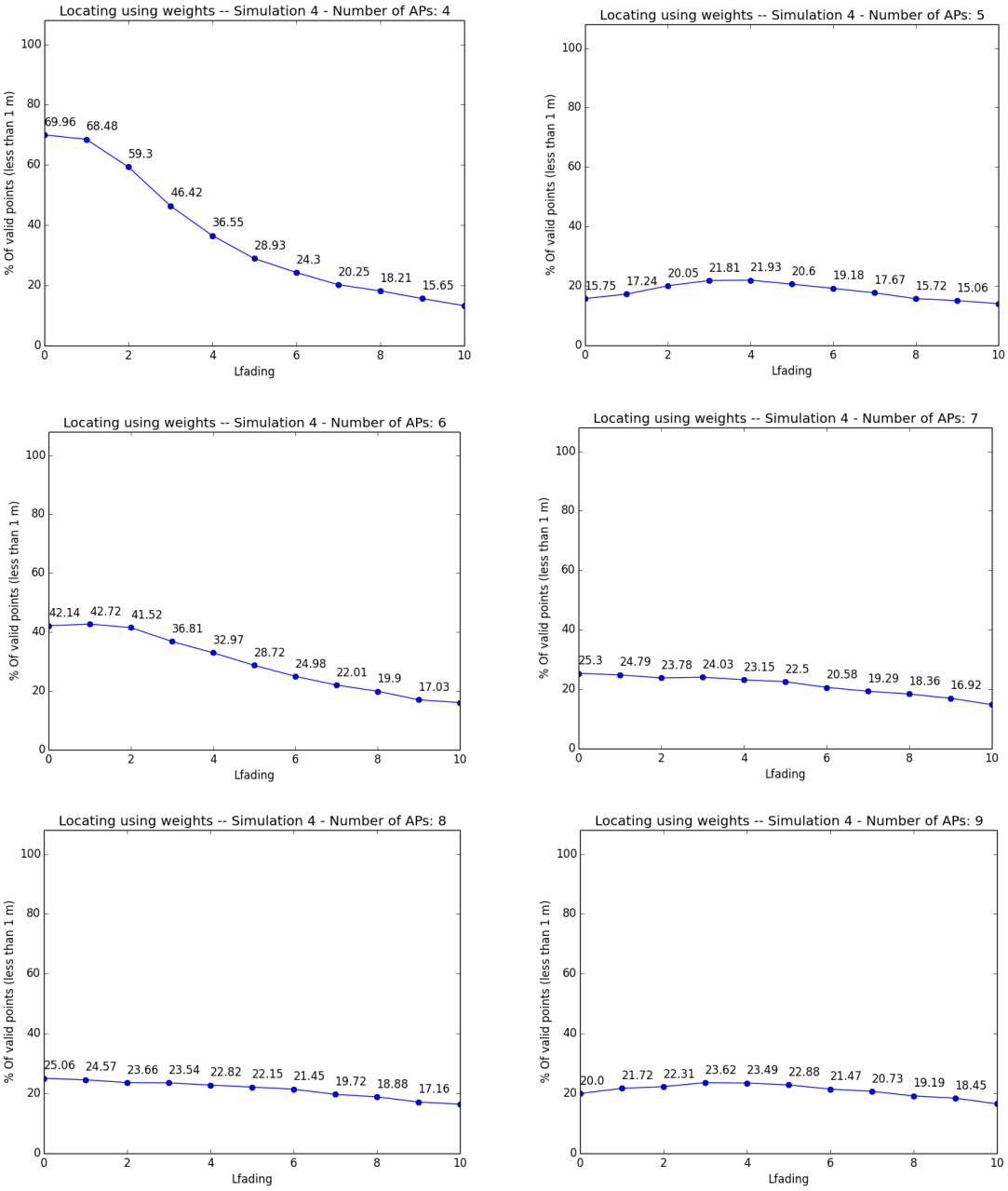


Figura 38. Ejecución de la simulación 4: Resultados para una simulación con un margen de error aceptable de 1 m.

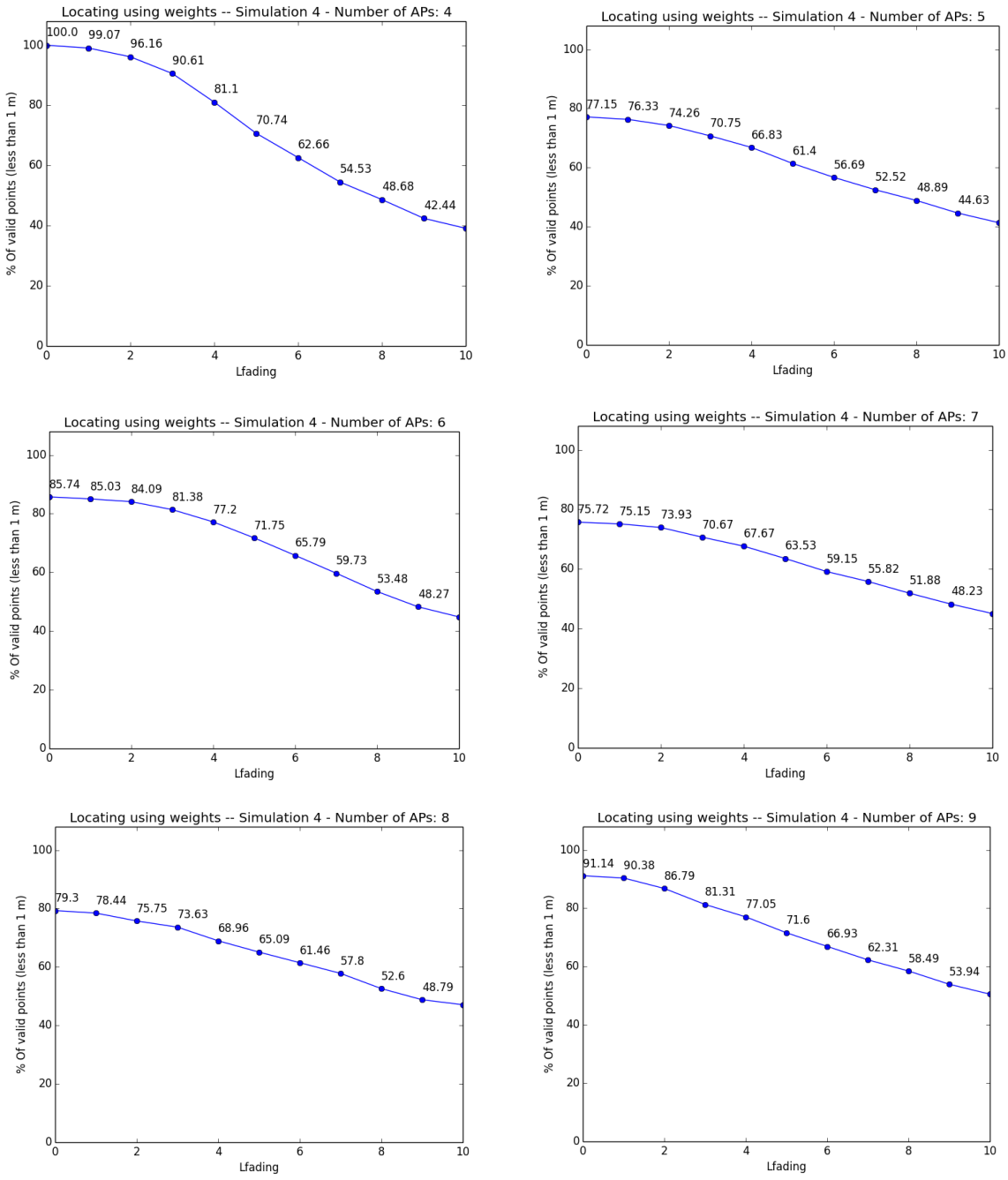


Figura 39. Ejecución de la simulación 4: Resultados para una simulación con un margen de error aceptable de 2 m.

Si analizamos los dos bloques de gráficas, se observa que la variable de fading afecta más a la simulación con un margen de error de 1 metro que a la de 2 ya que la pendiente de las curvas de la figura 39 son menos pronunciadas que las de la gráfica 38. Esto se debe a que el ruido que esta variable añade al RSSI no hace que sobrepase ese margen de error con tanta frecuencia como lo hace cuando el margen es menor.

6 CONCLUSIONES

Una vez finalizado este trabajo, es necesario detenerse y recopilar todas y cada una de las conclusiones que, durante el desarrollo de este proyecto, han ido surgiendo. Estas conclusiones deben estar basadas en los objetivos inicialmente propuestos, comparando los resultados obtenidos con los esperados. Por ello, en este capítulo se pretende resumir algunas de estas conclusiones, así como proponer varias ideas de desarrollo futuro que pueden mejorar el contenido y la capacidad de este proyecto, las cuales no han sido implementadas por no entrar dentro del alcance de este proyecto.

6.1 Conclusiones

Atendiendo a los objetivos inicialmente propuestos, podemos concluir que se han satisfecho la amplia mayoría, teniendo como primer propósito analizar los métodos de localización, una creciente tendencia en la evolución tecnológica de los últimos años.

Una vez que hemos sido capaces de lograr este primer objetivo, lo siguiente es poder asegurar la mayor escalabilidad posible de los métodos de localización, ya sea aumentando el número de elementos (STAs, APs…) o modificando los diferentes parámetros de las simulaciones que permiten obtener resultados en casos extremos.

Para ver en detalle lo anteriormente comentado, se detallarán las conclusiones para cada uno de los métodos descritos en este proyecto:

- **Tiempo de vuelo**: Se ha podido concluir según los resultados, que, conforme aumentamos el número de puntos de acceso el porcentaje de acierto va aumentando. Esto se debe a que cuantos más APs, el área de intersección resultante es menor, y poseemos más información desde otros puntos del mapa, siendo más fácil determinar su posición.

  Todo ello es posible teniendo en cuenta que según incrementamos el número de APs, el tiempo de ejecución del algoritmo aumenta notablemente.

  Además, si modificamos el índice que hace que el tiempo de procesado de la trama en el AP destino sea mayor o menor, en definitiva, el índice que hace que el tiempo de vuelo sea mayor o menor, el porcentaje de aciertos decrece o crece respectivamente, independientemente del número de APs.

  Este efecto que se acaba de detallar tiene menos impacto conforme aumentamos el número de APs. Se puede observar claramente si nos fijamos en la pendiente de las curvas de gráficas sucesivas.

- **Calibración**: El modelo que usamos para calibración, basado en el cálculo óptimo de los parámetros A y n, no generaliza bien cuando nos alejamos de las cercanías del punto de calibración. Esto se debe principalmente a que el parámetro n como exponente tiene un impacto enorme en los valores de señal calculados. Para una calibración perfecta deberíamos usar un modelo más complejo, con más parámetros y seguramente adaptado al entorno concreto (habría que hacer muchas medidas). Otra opción sería usar el modelo exacto propuesto por el IEEE en su estándar 802.11ax, que es el que hemos usado para simular la propagación durante todo este proyecto, y en cuyo caso el ajuste sería perfecto (usar el mismo modelo para propagar que para predecir). Sin embargo, este modelo del IEEE tampoco se cumple de forma exacta en la realidad, habiendo desviaciones de hasta 20 dB en el cálculo de la RSSI, lo cual puede traducirse en decenas o centenares de metros de error en el caso de bajas SNR. Por todo ello, este sistema es el menos recomendable para el cálculo de la posición de una STA,

por su poca robustez y capacidad de generalización a entornos reales.

- **Pesos:** Se ha comprobado que, la variable de desvanecimiento o fading, afecta más a las simulaciones con margen de error de 1 metro que a las de 2.

  Con menos APs y bajo fading, las simulaciones obtienen un porcentaje alto, pero con alto fading empeora mucho. Sin embargo, con más APs, empieza peor pero el aumento del fading no le afecta tanto.

6.2 Desarrollos futuros

En este apartado se pretenden incluir varias ideas de desarrollo futuro, de manera que se pueda continuar el trabajo aquí desarrollado. Además de los aspectos que se van a mencionar, se da por supuesto que existen muchas modificaciones que pueden mejorar la eficiencia y seguridad del sistema. Tómense como líneas de desarrollo principales, las siguientes:

- **Añadir más factores afecten al tiempo de vuelo**: Existen numerosos factores que hacen que el tiempo de vuelo tiempo de vuelo no sea tan trivial calcularlo. Por ejemplo, se pueden tener en cuenta el número de plantas que traspasa la señal WiFi, el número de paredes, obstáculos, personas en movimiento…
- **Implementar AoA**: debido a que los routers MIMO son bastante caros, quizás en el futuro bajen su coste y sea rentable implementar y utilizar el método de localización según el ángulo de llegada.
- **TDoA**: si nos es posible encontrar routers o tecnologías inalámbricas capaces de ofrecernos marcas de tiempos muy precisas, sería recomendable implementar el método de localización basado en la diferencia de tiempos de llegada.
- **Pesos**: no existen ecuaciones perfectas, y es por ello que hay un camino de mejoras completamente abierto. Existen combinaciones de fórmulas o ecuaciones por descubrir que hacen que el resultado obtenido sea más preciso que el obtenido en este proyecto.
- **Calibración**: en el apartado anterior se ha mencionado que para obtener mejores resultados deberíamos usar un modelo más complejo, con más parámetros y seguramente adaptado al entorno concreto (habría que hacer muchas medidas). Por tanto, nos encontramos ante una línea de mejora.
- **Algoritmo de trilateración**: como se ha detallado otros apartados, por ejemplo, en el de calibración, existen zonas las cuales el método de trilateración empleado no es capaz de arrojar los resultados correctamente. Es por ello que existe una dirección de mejora que consiste en perfeccionar este algoritmo.

Cabe decir que, además de las ideas mencionadas, pueden existir otras muchas ideas de mejoras, que podrían surgir cuando se llevase el sistema a un escenario de uso real y cotidiano.

REFERENCIAS

[1] Carlos Vialfa, «Introducción a wifi (802.11 o WiFi),» [En línea]. Available: https://es.ccm.net/contents/789-introduccion-a-wifi-802-11-o-wifi

[2] Sergio de Luz, «Wi-Fi Location: Qué es, cómo funciona y para qué sirve este estándar de geoposicionamiento en interiores con Wi-Fi,» [En línea]. Available: https://www.redeszone.net/2017/03/11/wi-fi-location-funciona-sirve-este-estandar-geoposicionamiento-interiores-wi-fi/

[3] Guido van Rossum, «Python,» [En línea]. Available: https://www.python.org

[4] Fidel Ramón García Pedraja, Vicente Quílez Sánchez, «IEEE 802.11(Wi-Fi) El estándar de facto para WLAN,» [En línea]. Available: https://www.coit.es/publicac/publbit/bit138/wifi.pdf

[5] Juan Carrasco Alonso, «Desarrollo de una herramienta para la localización de dispositivos en entornos

inalámbricos,» [En línea]. Available: https://repositorio.unican.es/xmlui/handle/10902/8797

[6] Keith Shaw, «Estándares y velocidades de Wi-Fi explicados y comparados,» [En línea]. Available: http://www.cwv.com.ve/estandares-y-velocidades-de-wi-fi-explicados-y-comparados/

[7] Keith Shaw, «802.11: estándares de Wi-Fi y velocidades,» [En línea]. Available: http://www.networkworld.es/wifi/80211-estandares-de-wifi-y-velocidades

[8] Python, «Tkinter27,» [En línea]. Available: https://docs.python.org/2/library/tkinter.html

[9] L. Mainetti, L. Patrono, y I. Sergi, «A survey on indoor positioning systems,» [En línea]. Available: http://dx.doi.org/10.1109/SOFTCOM.2014.7039067

[10] Jianqiao Xiong, Qin Qin, Kemin Zeng, «A Distance Measurement Wireless Localization Correction Algorithm Based On RSSI,» [En línea]. Available: https://ieeexplore.ieee.org/document/7081988/

[11] Nickhamy, «¿Qué es la trilateración?,» [En línea]. Available: http://empresiente.com/article/qu-es-la-trilateracin

[12] Noomrevlis, «Trilateration,» [En línea]. Available: https://github.com/noomrevlis/trilateration

[13] Eva María García, « Técnicas de Localización en Redes Inalámbricas de Sensores,» [En línea]. Available: https://www.researchgate.net/publication/228705728_Tecnicas_de_Localizacion_en_Redes_Inalambricas_de_Sensores

[14] Python, «Tkinter27,» [En línea]. Available: https://docs.python.org/2/library/tkinter.html

[15] Luis M. Gonzalez, «¿Cuáles son algunas de las limitaciones de Python?,» [En línea]. Available: https://es.quora.com/Cuáles-son-algunas-de-las-limitaciones-de-Python

[16] Oghenekome Oteri, Frank La Sita, RuiYang, Monisha Ghosh, Robert Olesen, «Improved Spatial Reuse for Dense 802.11 WLANs,» [En línea]. Available: https://zapdf.com/improved-spatial-reuse-for-dense-80211-wlans.html

ANEXO A: CARACTERÍSTICA DEL EQUIPO

En este Anexo se muestran las características del equipo empleado en la realización de este trabajo.

- Equipo de trabajo:
  - MacBook Pro (15-inch, 2017)
  - 2,8 GHz Intel Core i7
  - 16 GB 2133 MHz LPDDR3
  - Radeon Pro 555 2048 MB
  - Intel HD Graphics 630 1536 MB
  - macOS High Sierra 10.13.5
  - Python 2.7.10
  - Atom 1.30
  - Gitlab

ANEXO B: GUÍA DE EJECUCIÓN

En este apartado se propondrá una serie de pasos para una completa y correcta ejecución del proyecto.

- En primer lugar, abrimos la consola de comandos y accedemos al directorio TFG-JMND. Recuerda que Python debe estar instalado: *cd ./TFG-JMND*

- Ejecutamos el programa principal: *python main.py*, nos pedirá la posición del STA. Escribimos 500-500, por ejemplo.

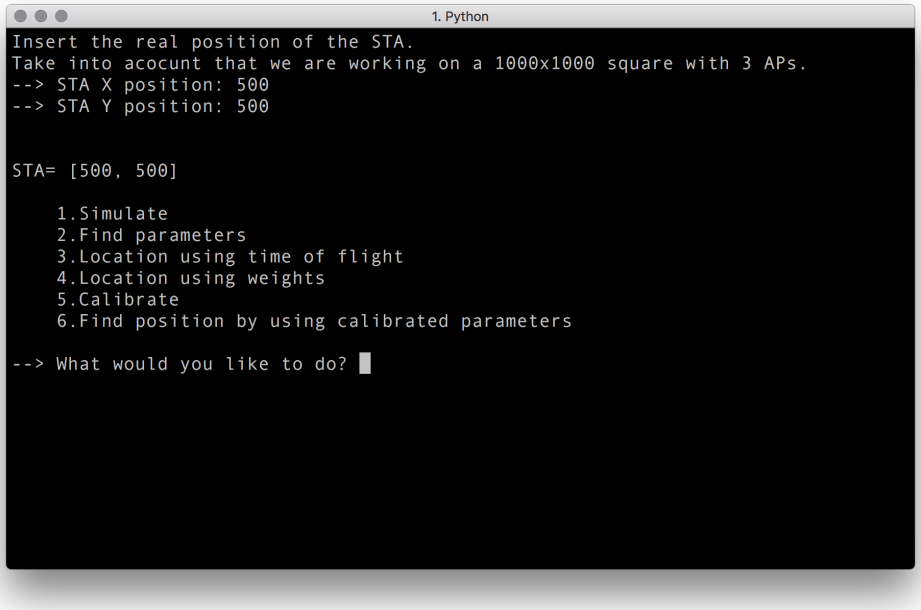


Figura 40. Ejecución main.py

- Es obligatorio realizar la opción 1 antes de hacer nada más.

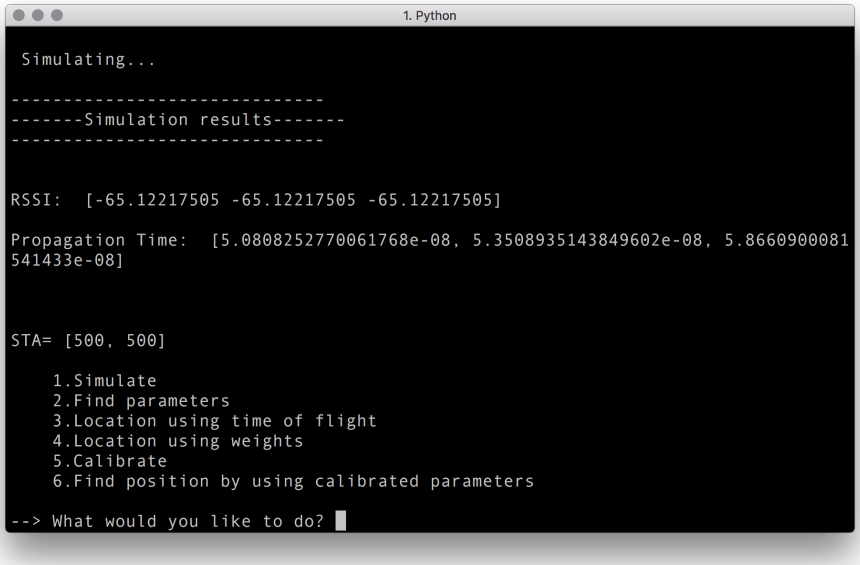


Figura 41. Resultado de la opción 1.

- Seguidamente, podemos ejecutar cualquiera de las opciones restantes (2-6), teniendo en cuenta que antes de ejecutar la 6 debemos haber realizado la 5 al menos una vez. Se muestran ejemplos de las ejecuciones de todas las opciones restantes.

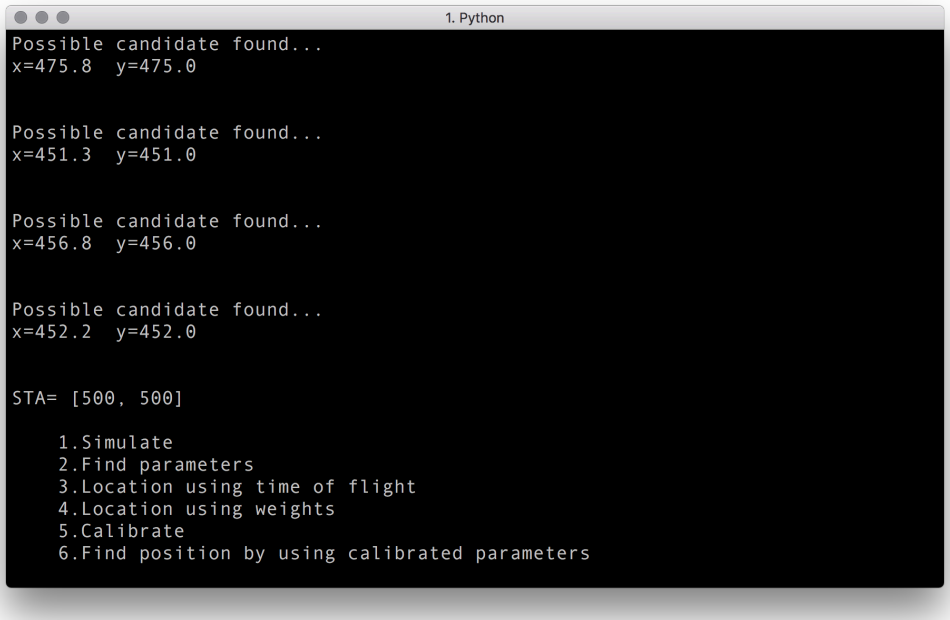


Figura 42. Resultado de la opción 2.

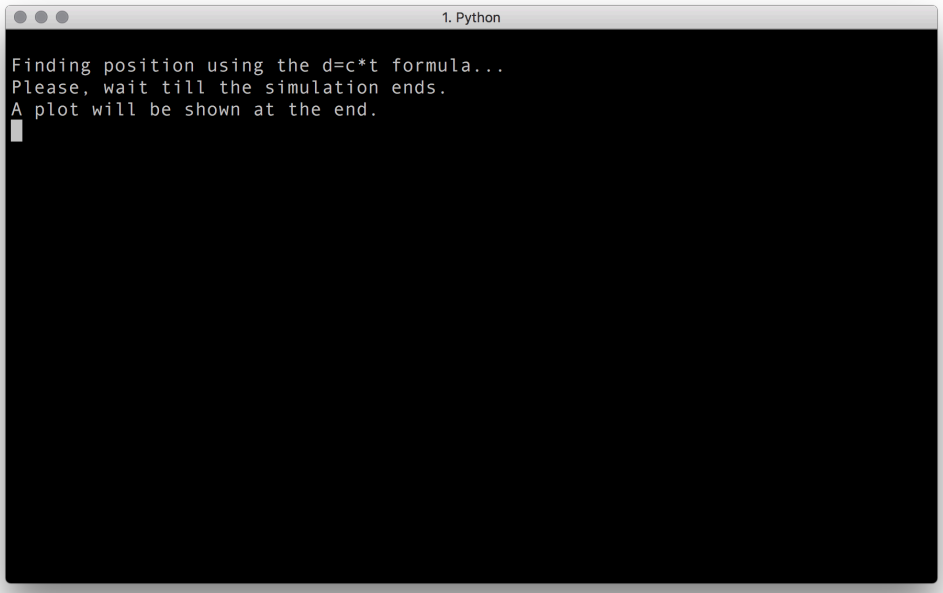


Figura 43. Resultado de la opción 3.

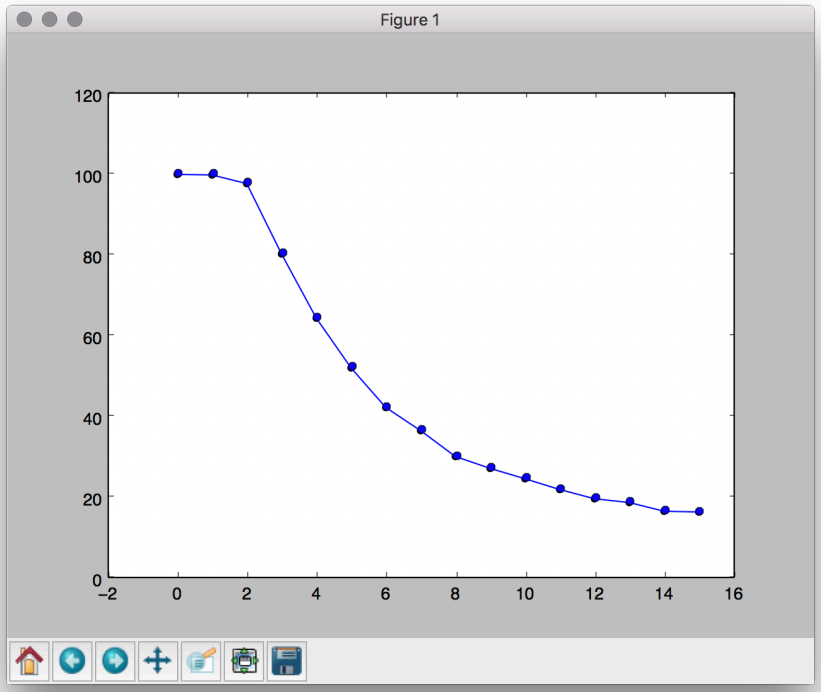


Figura 44. Gráfica mostrada al terminar la simulación de la opción 3.

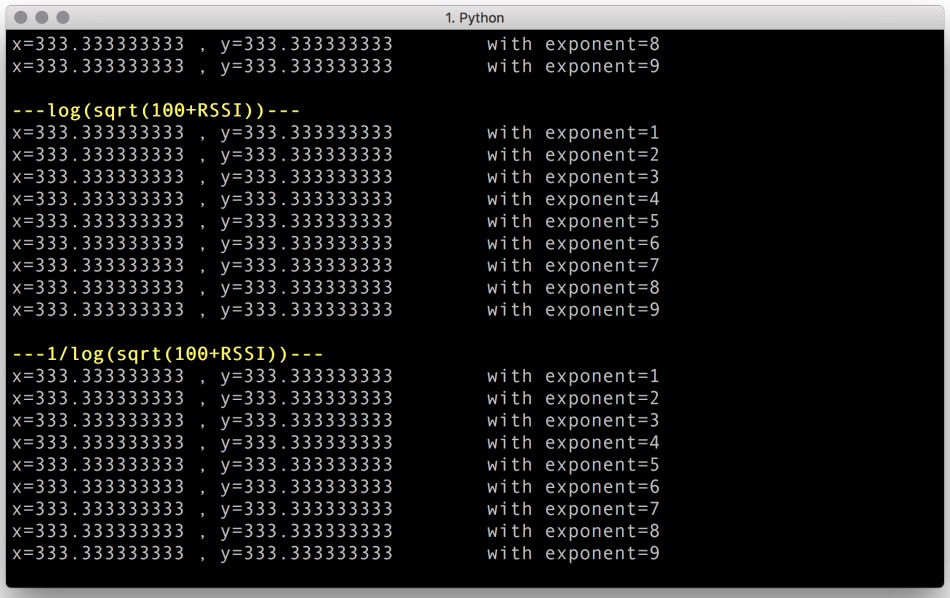


Figura 45. Resultado de la opción 4.

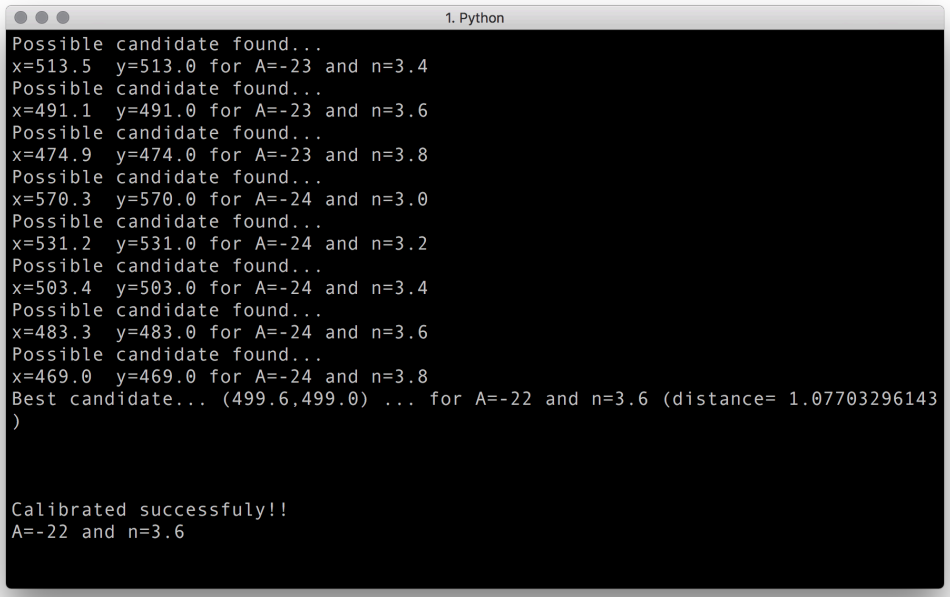


Figura 46. Resultado de la opción 5.

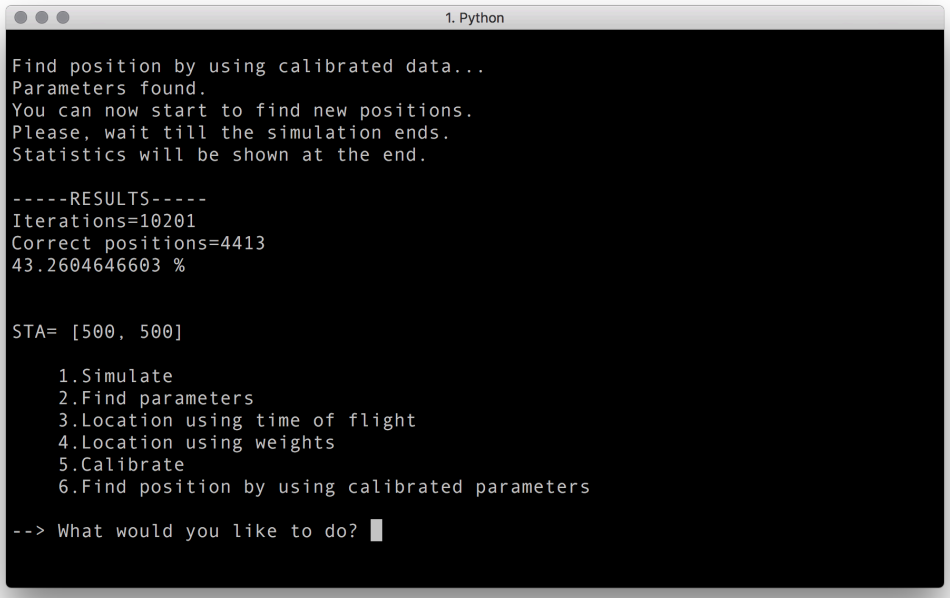


Figura 47. Resultado de la opción 6.

- También podemos ver los ficheros resultantes de la ejecución de cada simulación entrando en el directorio *results*.

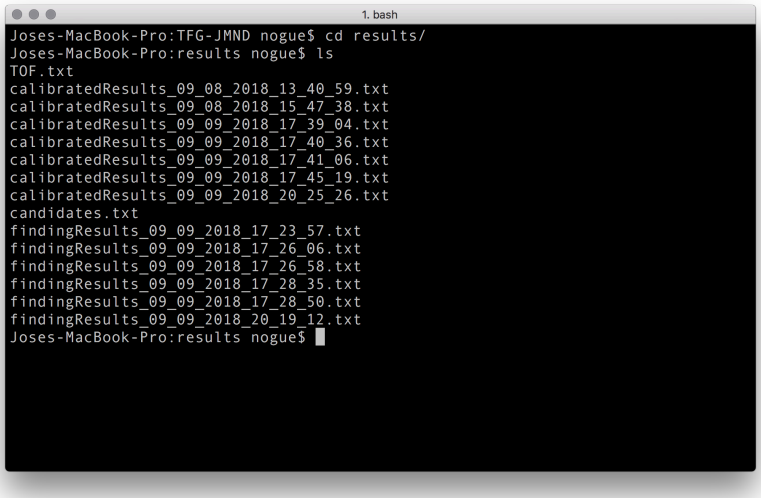


Figura 48. Listado de ficheros del directorio *results* del programa principal.

- Finalmente, para llevar a cabo las simulaciones, basta con entrar en el directorio de cada una de ellas

Simulation 1-4, y ejecutar el archivo *main.py*. Se pondrán ejemplos de la ejecución de cada simulación.

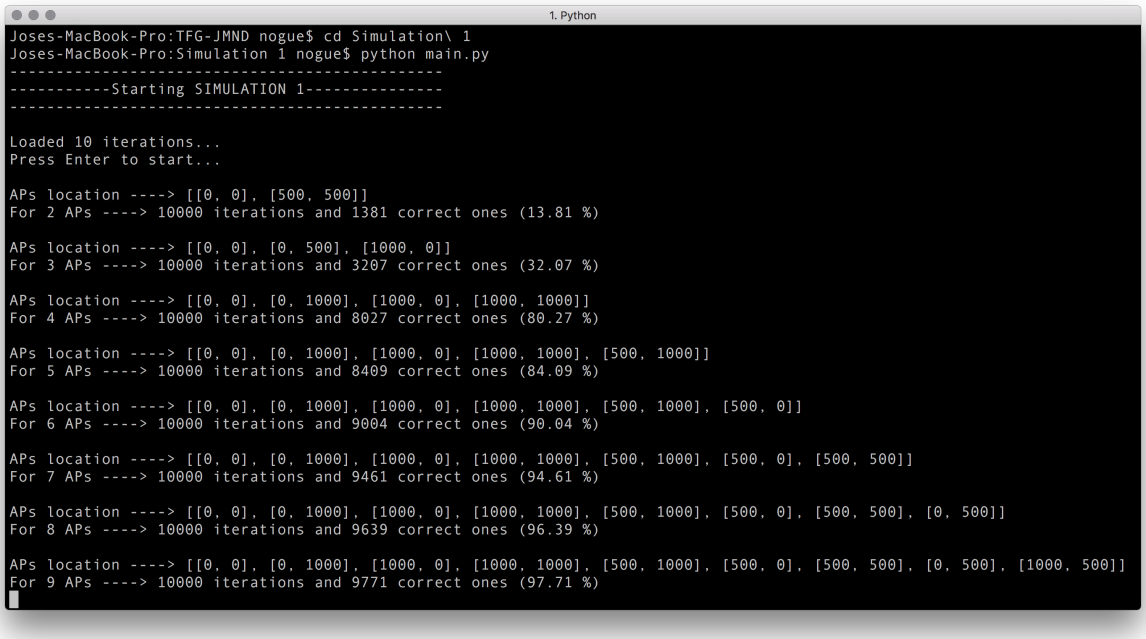


Figura 49. Resultado de la ejecución de la simulación 1.

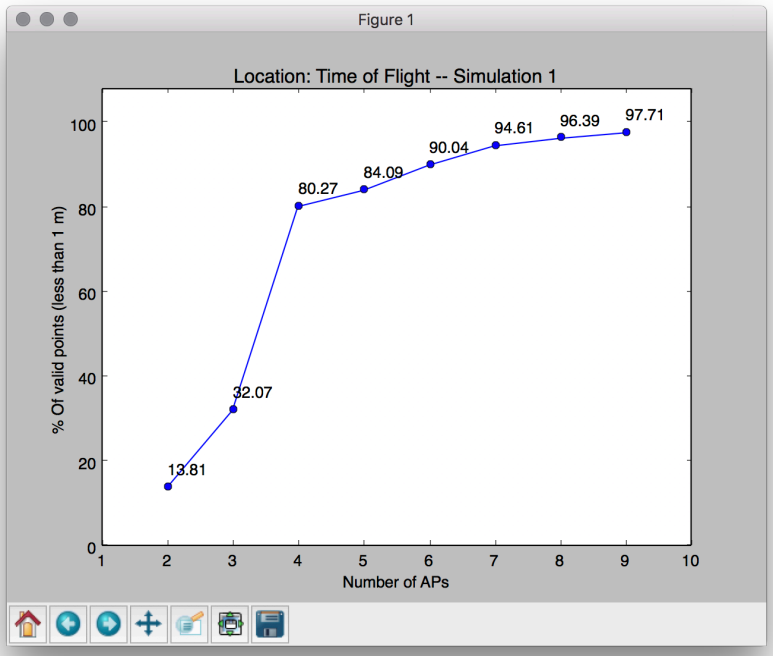


Figura 50. Gráfica resultante de la simulación 1.

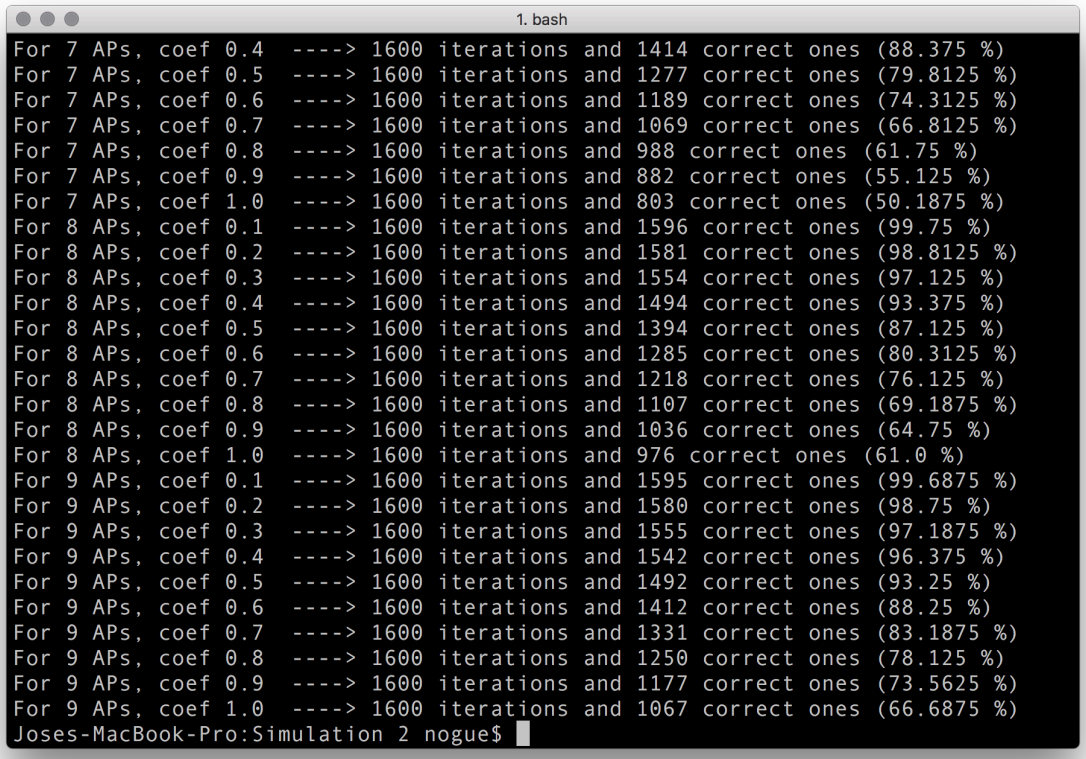


Figura 51. Resultado de la ejecución de la simulación 2.

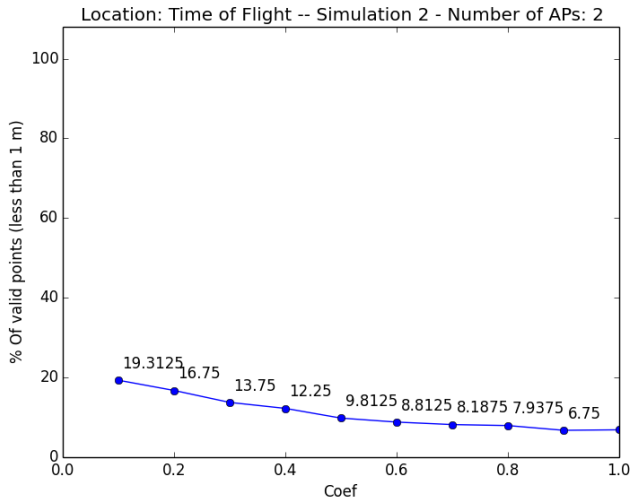


Figura 52. Ejemplo de una de las gráficas guardadas tras finalizar la simulación 2.

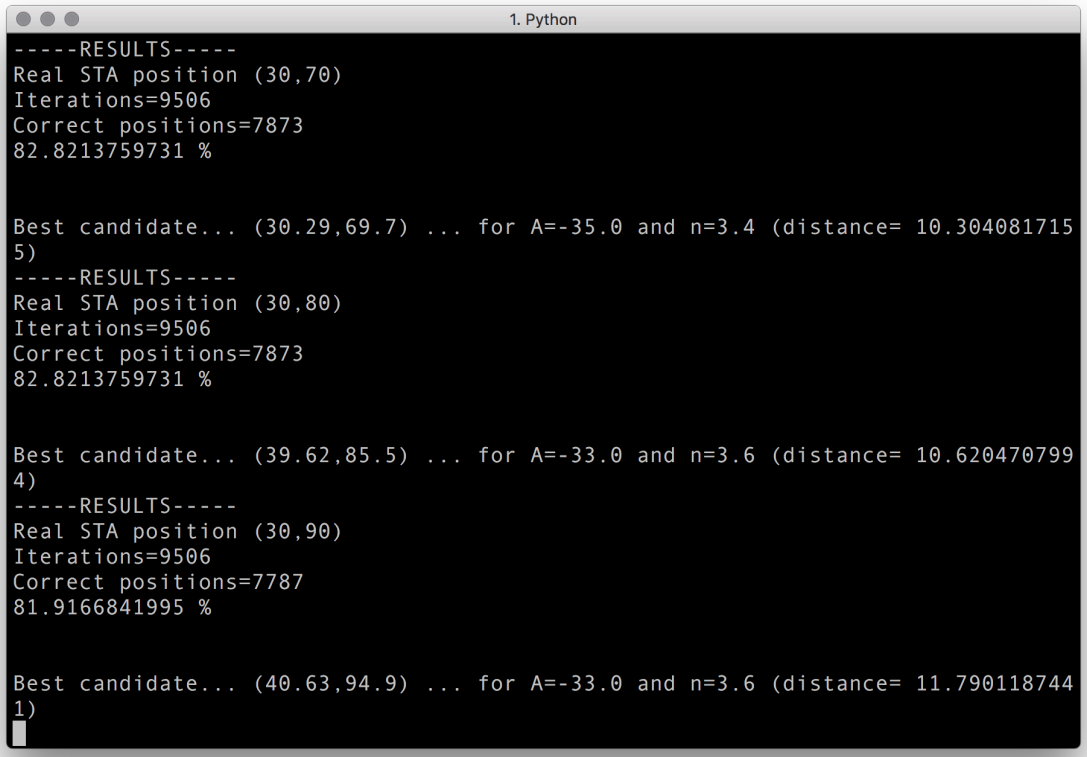


Figura 53. Resultado de la ejecución de la simulación 3.

- Podemos ver en el directorio *results* los resultados que se obtienen durante la simulación. La duración de esta simulación es de varias horas ya que hace un barrido muy minucioso del mapa de localización.

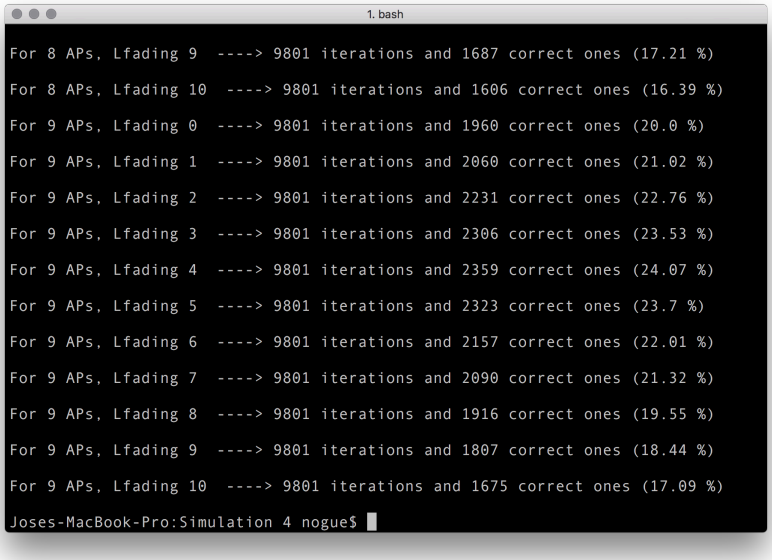


Figura 54. Resultado de la ejecución de la simulación 4.

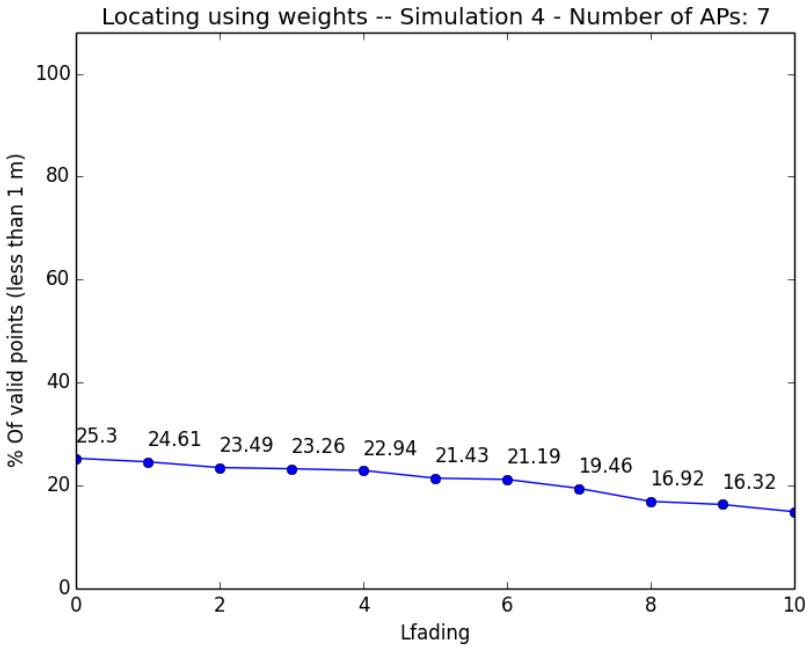


Figura 55. Gráfica de ejemplo guardadas en el directorio de la simulación 4 tras su finalización.

ANEXO C: CÓDIGOS

En este apartado se detallará el código empleado en la elaboración de este proyecto. En el apartado de las simulaciones se indicará el código de aquellos archivos que hayan cambiado significativamente con respecto al del programa original.

main.py

```
'''
-------------------- main.py --------------------
It shows the main menu
Author: Jose Manuel Noguerol Diaz
-------------------- main.py --------------------
'''

import propagation
import locationHandler as location
import sys
import os
import trilateration as trilaterate
import datetime
import matplotlib.pyplot as plt

# Clear the command line
os.system("clear")

# STA position
print "Insert the real position of the STA."
print "Take into acocunt that we are working on a 1000x1000 square with 3 APs."

apx=raw_input("--> STA X position: ")
apy=raw_input("--> STA Y position: ")
STA=[int(apx),int(apy)]

# APs positions
APs=[[0,0],[1000,0],[0,1000]]
#Frequency in Hz
f=[5e9,5e9,5e9]
#Tx power
Ptx=0


while True:
    print "\n\nSTA=",STA
    print ("""
    1.Simulate
    2.Find parameters
    3.Location using time of flight
    4.Location using weights
    5.Calibrate
    6.Find position by using calibrated parameters
    """)
    ans=raw_input("--> What would you like to do? ")
    if ans=="1":
        os.system("clear")
        print("\n Simulating...")

        RSSI,propagation_time=propagation.propagation(STA,APs,f,Ptx,0.3)
        print "\n------------------------------"
        print "-------Simulation results-------"
        print "------------------------------\n"
        print "\nRSSI: ",RSSI,"\n"
        print "Propagation Time: ",propagation_time,"\n"
```

```
elif ans=="2":
    os.system("clear")

    print "\n Finding parameters using the d=10^(A-RSSI)*10n formula..."
    print "----------------------"
    print "Trilateration results"
    print "----------------------"
    location.findParams(RSSI,APs,STA)

elif ans=="3":
    os.system("clear")
    print "\nFinding position using the d=c*t formula..."
    print "Please, wait till the simulation ends."
    print "A plot will be shown at the end."
    resultList=[]
    for coef in range(0,16,1):
        iterations=0
        correct=0
        #Sweeps the two axes
        for napx in range(0,501,5):
            for napy in range(0,501,5):
                iterations=iterations+1
                nSTA=[napx,napy]
                nRSSI,propagation_time=propagation.propagation(nSTA,APs,f,Ptx,coef/10.0)

                x,y,dist=location.distanceUsingTimeOfFlight(APs,propagation_time)

                file= open("./results/TOF.txt","a")
                file.write("x={0}  y={1} \t - {2}   {3} --- {4}" .format(x,y,napx,napy,dist))
                file.write("\n")
                file.close()
                if (x != -99 and y != -99):
                    op1= abs(x-napx)
                    op2= abs(y-napy)
                    if op1<100 and op2 <100:
                       correct=correct+1
                       file= open("./results/TOF.txt","a")
                       file.write("x={0}  y={1} \t - {2}   {3} ------> CORRECTO--- {4}" .format(x,y,napx,napy,dist))
                       file.write("\n")
                       file.close()

        result=100.0*correct/iterations

        resultList.append(result)
    plt.scatter(range(0,16,1),resultList)
    plt.plot(range(0,16,1),resultList,linestyle='-', marker='o')

    plt.show()

elif ans=="4":
    os.system("clear")
    print "\n Finding position using weights..."
    location.positionUsingWeights(APs,RSSI)
```

```
elif ans=="5":
    os.system("clear")
    A,n=0,0
    print "\nCalibrating..."
    [A,n]=location.calibrate(RSSI,APs,STA)
    if A!=0 and n!=0:
        print "\n\n\nCalibrated successfuly!!"
        print "A={0} and n={1}".format(A,n)
    else:
        print "Error calibrating..."

elif ans=="6":
    os.system("clear")
    iterations=0
    correct=0
    print("\nFind position by using calibrated data...")
    if A==0 and n==0:
        print "You must calibrate first!\n"
    else:
        print "Parameters found. \nYou can now start to find new positions."
        print "Please, wait till the simulation ends."
        print "Statistics will be shown at the end.\n"
        #Get the current date to print the datas to a file
        date=datetime.datetime.now()
        fileName="./results/calibratedResults_"+date.strftime("%m_%d_%Y_%H_%M_%S")+".txt"
        fi= open(fileName,"a")
        fi.write('STA (X)\t\t'+'STA (Y)\t\t'+'x\t\t'+'y')
        fi.write("\n")
        fi.close()

        iterations=0
        correct=0
        MARGIN=200
        #Sweeps the two axes
        for napx in range(0,501,5):
            for napy in range(0,501,5):
                iterations=iterations+1
                nSTA=[napx,napy]

                #Simulates for each position of the sweep
                nRSSI,propagation_time=propagation.propagation(nSTA,APs,f,Ptx,0.3)
                distances=[]

                for i in range(0,len(nRSSI)):
                    distances.append(pow(10,((   A-nRSSI[i]  ) / (  10*n  )    )))

                x,y=trilaterate.tri4TOF(APs,distances)
                if (x != -99 and y != -99):
                    op1= abs(x-napx)
                    op2= abs(y-napy)
                    if op1<MARGIN and op2 <MARGIN:
                        correct=correct+1
                        fi= open(fileName,"a")
                        string=str(napx)+'\t\t'+str(napy)+'\t\t'+str(x)+'\t\t'+str(y)+' CORRECT-------------->>>>>>>>>' + '\t\t'+ str(distances) + '\n'
                        fi.write(string)
                        fi.close()

                  else:
                      fi= open(fileName,"a")
                      string=str(napx)+'\t\t'+str(napy)+'\t\t'+str(x)+'\t\t'+str(y)+ '\t\t'+ str(distances) +'\n'
                      fi.write(string)
                      fi.close()

        print ("-----RESULTS-----")
        print "Iterations={0}".format(iterations)
        print "Correct positions={0}".format(correct)
        print "{0} %".format(100.0*correct/iterations)

    plt.show()

elif ans !="":
    os.system("clear")
    print("\n Not a valid choice!")
```

Figura 56. main.py

```
'''
---------------------- locationHandler.py ----------------------
Functions that let the user to use a locationb algorithm
Author: Jose Manuel Noguerol Diaz
---------------------- locationHandler.py ----------------------
'''

import math
import numpy
from sympy import Symbol
from sympy.solvers import solve
from sympy.functions import re
import time
import datetime
import trilateration
import os
import math

#Let us print with colours.
class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'

'''
-----findParams----
Used to handle the second option (Find parameter)
Change the parameters of the trilateration for each iteration
'''

def findParams(RSSI,APs,STA):
    d1=[]
    d2=[]
    d3=[]

    A=[]
    N=[]

    #Get the current date to print the datas to a file
    date=datetime.datetime.now()

    #Remove the candidate file if it already exists
    if(os.path.isfile("./results/candidates")):
        os.remove("./results/candidates")
    #Create the file in which the results will be typed.
    fileName="./results/findingResults_"+date.strftime("%m_%d_%Y_%H_%M_%S")+".txt"
    f= open(fileName,"a")
    f.write('A\t\t'+'N\t\t'+'x\t\t\t\t\t'+'y\t\t\t\t\t'+'t')
    f.write("\n")
    f.close()
```

```
    #Sweeps the A and n parameter
    for a in range(-100,-810,-10):
        for n in range(10,40,2):
            A.append(a)
            N.append(n)
            distances=[]
            for i in range(0,len(RSSI)):

                distances.append(pow(10,((a/10)-RSSI[i])/(n))) #10*n/10

            trilateration.tri4Simulate(APs,distances,a/10,n/10.0,fileName,STA)

'''
-----distanceUsingTimeOfFlight----
Used to handle the third option (Time of Flight)
Calculates the distances and trilaterates
'''

def distanceUsingTimeOfFlight(APs,times):

    distances=[]
    for t in times:
        distances.append(3e8*(t/2))

    x,y=trilateration.tri4TOF(APs,distances,False)

    return x,y,distances

'''
-----positionUsingWeights----
Used to handle the fourth option (Weights)
Calculates them by using different formulas
'''

def positionUsingWeights(APs,RSSI):

    total=sum(RSSI)
    N=len(RSSI)
    # for rssi in RSSI:
    #     weights.append(rssi/total)
    # print(sum(weights)) #=1

    print "Calculating position using:"


    print bcolors.WARNING + bcolors.BOLD + "\n---(100+RSSI)^[1...10]---" + bcolors.ENDC
    computePOW(RSSI,APs,"res=pow(100+r,p)")

    print bcolors.WARNING + bcolors.BOLD + "\n---(1/RSSI)^[1...10]---" + bcolors.ENDC
    computePOW(RSSI,APs,"res=pow(1/r,p)")

    print bcolors.WARNING + bcolors.BOLD + "\n---log(100+RSSI)---" + bcolors.ENDC
    computePOW(RSSI,APs,"res=pow(numpy.log10(100+r),p)")

    print bcolors.WARNING + bcolors.BOLD + "\n---X^RSSI---\n" + bcolors.ENDC
    for base in range(10,700,50):
        print "{0}^RSSI".format(base/100.0)
        computeEquation("res=pow(base,r)",RSSI,APs,base/100.0)

    print bcolors.WARNING + bcolors.BOLD + "\n---1/sqrt(100+RSSI)---" + bcolors.ENDC
    computePOW(RSSI,APs,"res=pow(1/numpy.sqrt(100+r),p)")

    print bcolors.WARNING + bcolors.BOLD + "\n---log(sqrt(100+RSSI))---" + bcolors.ENDC
    computePOW(RSSI,APs,"res=pow(numpy.log10(numpy.sqrt(100+r)),p)")

    print bcolors.WARNING + bcolors.BOLD + "\n---1/log(sqrt(100+RSSI))---" + bcolors.ENDC
    computePOW(RSSI,APs,"res=pow(1/numpy.log10(numpy.sqrt(100+r)),p)")
```

```
def computePOW(RSSI,APs,equation):
    N=len(RSSI)
    for p in range(1,10):
        beta=[]
        weights=[]
        betaTotal=0
        x,y=0,0
        for r in RSSI:
            exec(equation)
            beta.append(res)

        betaTotal=sum(beta)
        for b in beta:
            weights.append(b/betaTotal)

        #Calculates the final position
        for i in range(0,N):
            x=x+(weights[i]*APs[i][0])
        for j in range(0,N):
            y=y+(weights[i]*APs[j][1])

        print "x={0} , y={1} \t with exponent={2}".format(x,y,p)


'''
-----computeEquiation----
Used to make the code simpler
Let us to compute an equiation.
Base parameter can be used in the equation parameter as a changin variable
'''

def computeEquation(equation,RSSI,APs, base=None):
    N=len(RSSI)
    beta=[]
    weights=[]
    betaTotal=0
    x,y=0,0


    for r in RSSI:
        exec(equation)
        beta.append(res)

    betaTotal=sum(beta)
    for b in beta:
        weights.append(b/betaTotal)

    #Calculates the final position
    for i in range(0,N):
        x=x+(weights[i]*APs[i][0])
    for j in range(0,N):
        y=y+(weights[j]*APs[j][1])

    print "x={0} , y={1}".format(x,y)
```

```
'''
-----calibrate----
Used to handle the fifth option (Calibrate)
Change the variables and trilaterates to find the right values.
'''

def calibrate(RSSI,APs,STA):

    A=[]
    N=[]
    xList=[]
    yList=[]
    estimatedA=0
    estimatedN=0
    bestA=0
    bestN=0
    for a in range(250,-250,-10):
        for n in range(20,40,2):

            distances=[]
            for i in range(0,len(RSSI)):

                distances.append(pow(10,((a/10)-RSSI[i])/(n))) #10*n/10

            valid,xx,yy=trilateration.validate(APs,distances,a/10,n/10.0,STA)

            #If the point is valid all the parameters are saved
            if valid:
                A.append(a/10)
                N.append(n/10.0)
                xList.append(xx)
                yList.append(yy)

    if not xList or not yList:
        print "ERROR FINDING VALID POINTS"
    else:
        dist=[]
        yIndex=0
        for x in xList:
            dist.append(pow(STA[0]-x,2) + pow(STA[1]-yList[yIndex],2))
            yIndex=yIndex+1
        bestIndex=dist.index( min(dist) )
        bestX=xList[bestIndex]
        bestY=yList[bestIndex]
        bestA=A[bestIndex]
        bestN=N[bestIndex]

        print "Best candidate... ({0},{1}) ... for A={2} and n={3} (distance= {4})".format(bestX,bestY,bestA,bestN,numpy.sqrt(dist[bestIndex])

    return bestA,bestN
```

Figura 57. locationHandler.py

```
'''
--------------------- propagation.py -------------------------
Let us get RSSI and propagation time based on an 802.11
pathloss model
Author: Jose Manuel Noguerol Diaz
--------------------- propagation.py -------------------------
'''

import numpy as np
# Esta funcion simula la propagacion de la senal WiFi en una habitacion
# grande. Los APs sondean a la STA y tras cierto tiempo (propagation_time)
# reciben una respuesta con cierta potencia (RSSI)

# % INPUTS ----------------------------------------------
# % STA: vector 2x1 con la posicion de STA.
# % APs: matrix de 2xN con la posicion de los N APs.
# % f: vector de 1xN con las frecuencias de cada AP en Hz.
# % Ptx: Potencia transmitida en dBm.
#
# % OUTPUTS ----------------------------------------------
# % RSSI: vector 1xN con las RSSI recibidas en los APs.
# % propagation_time: vector con los tiempos de ida y vuelta del mensaje que manda el AP.

def propagation (STA,APs,f,Ptx,coef):
    scale=100
    c=3e8
    Lfading=0
    Lwalls=0
    N=len(APs)
    direction_vectors=[]
    distance=[]
    propagation_time=[]
    d=[]
    L=[]
    for i in range(0,N):
        direction_vectors.append(np.subtract(STA,APs[i]))
        distance.append(np.linalg.norm(direction_vectors[i])/scale)
        normal=np.random.random()
        propagation_time.append(2*(1+coef*normal)*distance[i]/c)
        d.append(max(distance[i],1))
        if (d[i]>5) > 0:
            L.append(40.05+20*(np.log10(f[i]/1e9)/2.4)+20*np.log10(min(d[i],5))+(d[i]>5)*35*np.log10(d[i]/5))
        else:
            L.append(40.05+20*(np.log10(f[i]/1e9)/2.4)+20*np.log10(min(d[i],5)))

    L_space=(np.add(L,Lwalls))
    RSSI=(np.subtract(np.subtract(Ptx,L_space),Lfading*np.random.normal(0,1,N)))
    return RSSI,propagation_time
```

Figura 58. propagation.py

```
'''
---------------------- trilateration.py ----------------------
Trilateration algorithm or functions that use it
Author: Jose Manuel Noguerol Diaz
---------------------- trilateration.py ----------------------
'''

import json
import math
from json import encoder
import time
import re
import matplotlib.pyplot as plt
import datetime


encoder.FLOAT_REPR = lambda o: format(o, '.2f')

class base_station(object):
    def __init__(self, lat, lon, dist):
        self.lat = lat
        self.lon = lon
        self.dist = dist

class point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

class circle(object):
    def __init__(self, point, radius):
        self.center = point
        self.radius = radius

class json_data(object):
    def __init__(self, circles, inner_points, center):
        self.circles = circles
        self.inner_points = inner_points
        self.center = center

def serialize_instance(obj):
    d = {}
    d.update(vars(obj))
    return d

def get_two_points_distance(p1, p2):
    return math.sqrt(pow((p1.x - p2.x), 2) + pow((p1.y - p2.y), 2))
```

```
def get_two_circles_intersecting_points(c1, c2):
    p1 = c1.center
    p2 = c2.center
    r1 = c1.radius
    r2 = c2.radius

    d = get_two_points_distance(p1, p2)
    # if to far away, or self contained - can't be done
    if d >= (r1 + r2) or d <= math.fabs(r1 -r2):
        return None

    a = (pow(r1, 2) - pow(r2, 2) + pow(d, 2)) / (2*d)
    h  = math.sqrt(pow(r1, 2) - pow(a, 2))
    x0 = p1.x + a*(p2.x - p1.x)/d
    y0 = p1.y + a*(p2.y - p1.y)/d
    rx = -(p2.y - p1.y) * (h/d)
    ry = -(p2.x - p1.x) * (h / d)
    return [point(x0+rx, y0-ry), point(x0-rx, y0+ry)]

def get_all_intersecting_points(circles):
    points = []
    num = len(circles)
    for i in range(num):
        j = i + 1
        for k in range(j, num):
            res = get_two_circles_intersecting_points(circles[i], circles[k])
            if res:
                points.extend(res)
    return points

def is_contained_in_circles(point, circles):
    for i in range(len(circles)):
        if (get_two_points_distance(point, circles[i].center) > (circles[i].radius) +10): #+10 stands for a margin of erro
            return False
    return True

def get_polygon_center(points):
    center = point(0, 0)
    num = len(points)
    if (num!=0):
        for i in range(num):
            center.x += points[i].x
            center.y += points[i].y
        center.x /= num
        center.y /= num
    return center
```

```
'''
-----tri4Simulate----
Used to trilaterate for the second option (Find parameter)
Write down the result (x,y point) to a file
If the calculated point is between the range built by using the error parameter,
it is also written in the candidate file.
'''
def tri4Simulate(APs,distances,A,N,fileName,STA):

    x1=APs[0][0]
    y1=APs[0][1]
    x2=APs[1][0]
    y2=APs[1][1]
    x3=APs[2][0]
    y3=APs[2][1]

    #Error margin
    margin = 70

    R1=distances[0]*100
    R2=distances[1]*100
    R3=distances[2]*100

    p1 = point(x1, y1)
    p2 = point(x2, y2)
    p3 = point(x3, y3)

    c1 = circle(p1, R1)
    c2 = circle(p2, R2)
    c3 = circle(p3, R3)

    circle_list = [c1, c2,c3]

    inner_points = []
    for p in get_all_intersecting_points(circle_list):
        if is_contained_in_circles(p, circle_list):
            inner_points.append(p)

    center = get_polygon_center(inner_points)
    in_json = json_data([c1, c2,c3], [p1, p2,p3], center)

    out_json = json.dumps(in_json, sort_keys=True,
                    indent=4, default=serialize_instance)

    # with open("data.json", 'w') as fw:
    #     fw.write(out_json)
    # fw.close();
    # m = re.search('[0-9.0-9]',out_json)

    pre=out_json[1:60]
    #Strings which contain the value of the position and invalid characters
    xxx=pre[23:35]
    yyy=pre[46:55]
    #Shorter strings which will contain the value of the position and (less) invalid characters
    xx,yy=[],[]
    #Default values for the position to check if there was error extracting the values
    x,y=-99,-99
```

```
    if (not '-' in xxx) or (not '-' in yyy): #no negative numbers

        for i in xxx:
            if i.isdigit() or i == ".": #Is a digit or contains a '.'
                xx.append(i)
        for j in yyy:
            if j.isdigit() or j == ".":
                yy.append(j)
        if len(xx)==0 or not len(yy)==0:
            #Contain the final value of the position if any digit has been appended
            x=float(''.join(xx))
            y=float(''.join(yy))
            f=open(fileName,"a")
            string=str(A)+'\t\t'+str(N)+'\t\t'+str(x)+'\t\t\t\t'+str(y)+'\t\t\t\t'+'\n'
            f.write(string)
            f.close()

            if (x<STA[0]+margin and x>STA[0]-margin) and (y<STA[1]+margin and y>STA[1]-margin):
                f= open("./results/candidates.txt","a")
                f.write("x={0}  y={1}  for A={2} and n={3}" .format(x,y,A,N))
                f.write("\n")
                f.close()
                print("\n\nPossible candidate found...")
                print "x={0}  y={1}" .format(x,y)

'''
-----tri4TOF----
Used to trilaterate for the third option (Time of Flight) and sixth
If True is given to the third parameter, a plot file will be created with the results.
'''
def tri4TOF(APs,distances, show = None):


    x1=APs[0][0]
    y1=APs[0][1]
    x2=APs[1][0]
    y2=APs[1][1]
    x3=APs[2][0]
    y3=APs[2][1]

    R1=distances[0]*100
    R2=distances[1]*100
    R3=distances[2]*100

    p1 = point(x1, y1)
    p2 = point(x2, y2)
    p3 = point(x3, y3)

    c1 = circle(p1, R1)
    c2 = circle(p2, R2)
    c3 = circle(p3, R3)

    circle_list = [c1, c2, c3]

    inner_points = []
    for p in get_all_intersecting_points(circle_list):
        if is_contained_in_circles(p, circle_list):
            inner_points.append(p)
```

```
center = get_polygon_center(inner_points)
in_json = json_data([c1, c2, c3], [p1, p2, p3], center)

out_json = json.dumps(in_json, sort_keys=True,
                indent=4, default=serialize_instance)

pre=out_json[1:60]
#Strings which contain the value of the position and invalid characters
xxx=pre[23:35]
yyy=pre[46:55]
#Shorter strings which will contain the value of the position and (less) invalid characters
xx,yy=[],[]

#Default values for the position to check if there was error extracting the values
x,y=-99,-99

if (not '-' in xxx) or (not '-' in yyy): #no negative numbers

    for i in xxx:
        if i.isdigit() or i == ".": #Is a digit or contains a '.'
            xx.append(i)
    for j in yyy:
        if j.isdigit() or j == ".":
            yy.append(j)
    if len(xx)==0 or not len(yy)==0:
        #Contain the final value of the position if any digit has been appended
        x=float(''.join(xx))
        y=float(''.join(yy))
        if show == True:
            print " x={0} , y={1} ".format(x,y)

            #Plot the circles
            circle1 = plt.Circle((x1, y1), R1, color='r', fill=False)
            circle2 = plt.Circle((x2, y2), R2, color='blue', fill=False)
            circle3 = plt.Circle((x3, y3), R3, color='black', clip_on=False,fill=False)

            fig, ax = plt.subplots()
            ax.set_xlim((-max(R1*1.3,R2*1.3), max(R1*1.3,R2*1.3)))
            ax.set_ylim((-max(R1*1.3,R2*1.3), max(R1*1.3,R2*1.3)))

            point1=plt.plot(x1,y1,marker='o', markersize=8, color="red")
            point2=plt.plot(x2,y2,marker='o', markersize=8, color="blue")
            point3=plt.plot(x3,y3,marker='o', markersize=8, color="black")
            point4=plt.plot(x,y,marker='x', markersize=8, color="green")

            ax.add_artist(circle1)
            ax.add_artist(circle2)
            ax.add_artist(circle3)


            #Get the current date to print the datas to a file
            date=datetime.datetime.now()
            fig.savefig("plotcircles_"+date.strftime("%m_%d_%Y_%H_%M_%S")+".png")
return x,y
```

```
'''
-----validate----
Used to trilaterate for the option number 5 (Calibrate)
Returns if a point, and if it is valid
'''
def validate(APs,distances,A,N,STA):

    valid=False

    x1=APs[0][0]
    y1=APs[0][1]
    x2=APs[1][0]
    y2=APs[1][1]
    x3=APs[2][0]
    y3=APs[2][1]

    #Error RANGE lets you to make a ceratin point valid or not
    RANGE=100

    R1=distances[0]*100
    R2=distances[1]*100
    R3=distances[2]*100

    p1 = point(x1, y1)
    p2 = point(x2, y2)
    p3 = point(x3, y3)

    c1 = circle(p1, R1)
    c2 = circle(p2, R2)
    c3 = circle(p3, R3)
    x=0
    y=0
    circle_list = [c1, c2,c3]

    inner_points = []
    for p in get_all_intersecting_points(circle_list):
        if is_contained_in_circles(p, circle_list):
            inner_points.append(p)

    center = get_polygon_center(inner_points)
    in_json = json_data([c1, c2,c3], [p1, p2,p3], center)

    out_json = json.dumps(in_json, sort_keys=True,
                      indent=4, default=serialize_instance)
    pre=out_json[1:60]
    #Strings which contain the value of the position and invalid characters
    xxx=pre[23:35]
    yyy=pre[46:55]
    #Shorter strings which will contain the value of the position and (less) invalid characters
    xx,yy=[],[]
    #Default values for the position to check if there was error extracting the values
    x,y=-99,-99
```

```
if (not '-' in xxx) or (not '-' in yyy): #no negative numbers

    for i in xxx:
        if i.isdigit() or i == ".": #Is a digit or contains a '.'
            xx.append(i)
    for j in yyy:
        if j.isdigit() or j == ".":
            yy.append(j)
    if len(xx)==0 or not len(yy)==0:
        #Contain the final value of the position if any digit has been appended
        x=float(''.join(xx))
        y=float(''.join(yy))

    if (x<STA[0]+RANGE and x>STA[0]-RANGE) and (y<STA[1]+RANGE and y>STA[1]-RANGE):
        print("Possible candidate found...")
        print "x={0}  y={1} for A={2} and n={3}" .format(x,y,A,N)
        valid=True

return valid,x,y
```

Figura 59. trilateration.py

Simulation 1

```
'''
---------------------- main.py ----------------------
It shows the main menu
Author: Jose Manuel Noguerol Diaz
---------------------- main.py ----------------------
'''

import propagation
import locationHandler as location
import sys
import os
import trilateration as trilaterate
import datetime
import matplotlib.pyplot as plt
import numpy as np

# Clear the command line
os.system("clear")
#Tx power
Ptx=0
#error margin allowed
MARGIN=100

print "------------------------------------------------"
print "-----------Starting SIMULATION 1---------------"
print "------------------------------------------------"

fp=open('./coordinates.txt', 'r')
line = fp.readline()
cnt = 1
APsList=[]

while line:
    APsList.append(line.strip())
    line = fp.readline()
    cnt += 1
fp.close()

print "\nLoaded {} iterations...".format(cnt)

foo=raw_input("Press Enter to start...")

#Arrays to append the results and the number of APs which it has been performed
numberList=[]
resultList=[]

for nAPs in range(2,10,1):

    #frecuencies array
    f=[]
    f=5e9*np.ones((nAPs,), dtype=int)

    iterations=0
    correct=0
    result=0
```

```
    #Sweeps the two axes
    for napx in range(1,1001,10):
        for napy in range(1,1001,10):

            iterations=iterations+1
            nSTA=[napx,napy]
            x=-99
            y=-99

            APs=location.parser(APsList[nAPs-1])

            nRSSI,propagation_time=propagation.propagation(nSTA,APs,f,Ptx,0.3)

            x,y=location.distanceUsingTimeOfFlight(APs,propagation_time)

            if (x != -99 and y != -99):
                op1= abs(x-napx)
                op2= abs(y-napy)
                if op1<MARGIN and op2 <MARGIN:
                   correct=correct+1

    result=100.0*correct/iterations
    resultList.append(result)
    numberList.append(nAPs)

    print "\nAPs location ----> {}".format(APs)
    print "For {} APs ----> {} iterations and {} correct ones ({} %)".format(nAPs,iterations,correct,result)

plt.title('Location: Time of Flight -- Simulation 1')
plt.xlabel('Number of APs')
plt.ylabel('% Of valid points (less than 1 m) ')
axes = plt.gca()
axes.set_xlim([1,10])
axes.set_ylim([0,108])

plt.plot(numberList,resultList,linestyle='-', marker='o')

#Print the value of each point
for i,j in zip(numberList,resultList):
    axes.annotate(str(j),xy=(i+0.01,j+3))

plt.show()
```

Figura 60. Simulation 1: main.py

```
'''
---------------------- locationHandler.py ----------------------
Functions that let the user to use a location algorithm
Author: Jose Manuel Noguerol Diaz
---------------------- locationHandler.py ----------------------
'''

import math
import numpy
from sympy import Symbol
from sympy.solvers import solve
from sympy.functions import re
import time
import datetime
import trilateration
import os
import math

'''
-----distanceUsingTimeOfFlight----
Used to handle the third option (Time of Flight)
Calculates the distances and trilaterates
'''

def distanceUsingTimeOfFlight(APs,times):

    distances=[]
    for t in times:
        distances.append(3e8*(t/2))

    x,y=trilateration.tri4TOF(APs,distances,False)

    return x,y

def parser (APsList):

    split=splitString(APsList)
    if len(APsList)>2:
        deleted=delete_(split)
    APs=buildString(deleted)
    return APs


def splitString(string):

    # Split the string based on space delimiter
    list_string = string.split(' ')

    return list_string

def delete_(string):
    deleted=[]

    for item in string:
        if item !='-':
            deleted.append(item)
    return deleted


def buildString(string):
    N=len(string)
    APs=[]
    for item in range(0,N,2):
        APs.append([int(string[item]),int(string[item+1])])
    return APs
```

Figura 61. Simulation 2: locationHandler.py

Simulation 2

```
'''
--------------------- main.py ---------------------
It shows the main menu
Author: Jose Manuel Noguerol Diaz
--------------------- main.py ---------------------
'''

import propagation
import locationHandler as location
import sys
import os,re
import trilateration as trilaterate
import datetime
import matplotlib.pyplot as plt
import numpy as np

# Clear the command line
os.system("clear")

Ptx=0
#error margin allowed
MARGIN=100

print "-------------------------------------------------"
print "-----------Starting SIMULATION 2---------------"
print "-------------------------------------------------"

fp=open('./coordinates.txt', 'r')
line = fp.readline()
cnt = 1
APsList=[]

while line:
    APsList.append(line.strip())
    line = fp.readline()
    cnt += 1
fp.close()

# Get current working directory
dir=os.getcwd()
pattern=".png"
for ff in os.listdir(dir):
    if re.search(pattern, ff):
        os.remove(os.path.join(dir, ff))
print "All previous PNG files have been deleted"

print "\nLoaded {} iterations...".format(cnt)

foo=raw_input("Press Enter to start...")
```

```
for nAPs in range(2,10,1):

    f=[]
    f=5e9*np.ones((nAPs,), dtype=int)

    coefList=[]
    resultList=[]
    for coef in range(1,11):
        iterations=0
        correct=0
        result=0

        #Sweeps the two axes
        for napx in range(1,1001,25):
            for napy in range(1,1001,25):

                iterations=iterations+1
                nSTA=[napx,napy]
                x=-99
                y=-99

                APs=location.parser(APsList[nAPs-1])
                # APs location won't be printed just to make the command line clearer.

                nRSSI,propagation_time=propagation.propagation(nSTA,APs,f,Ptx,coef/10.0)

                x,y=location.distanceUsingTimeOfFlight(APs,propagation_time)

                if (x != -99 and y != -99):
                    op1= abs(x-napx)
                    op2= abs(y-napy)
                    if op1<MARGIN and op2 <MARGIN:
                       correct=correct+1

        result=100.0*correct/iterations
        resultList.append(result)
        coefList.append(coef/10.0)

        print "For {} APs, coef {}  ----> {} iterations and {} correct ones ({} %)".format(nAPs,coef/10.0,iterations,correct,result)

    fig, ax = plt.subplots()
    axes = plt.gca()
    axes.set_xlim([0,1])
    axes.set_ylim([0,108])
    plt.plot(coefList,resultList,linestyle='-', marker='o')

    #Print the value of each point
    for i,j in zip(coefList,resultList):
        axes.annotate(str(j),xy=(i+0.01,j+3))

    axes.set_title('Location: Time of Flight -- Simulation 2 - Number of APs: '+str(nAPs))
    axes.set_xlabel('Coef')
    axes.set_ylabel('% Of valid points (less than 1 m) ')
    fig.savefig("Sim 2 "+"and "+str(nAPs)+" APs.png")
```

Figura 62. Simulation 2: main.py

Simulation 3

```
'''
--------------------- main.py ---------------------
It shows the main menu
Author: Jose Manuel Noguerol Diaz
--------------------- main.py ---------------------
'''

import propagation
import locationHandler as location
import sys
import os
import trilateration as trilaterate
import datetime
import matplotlib.pyplot as plt
import numpy as np

# Clear the command line
os.system("clear")

# APs positions
APs=[[0,0],[1000,0],[0,1000]]
#Frequency in Hz
f=[5e9,5e9,5e9]
#Tx power
Ptx=0
#Error allowed
error=1.35
#Get the current date to print the datas to a file
date=datetime.datetime.now()
fileName="./results/calibratedResults"+date.strftime("%m_%d_%Y_%H_%M_%S")+".txt"
fi= open(fileName,"a")


print "------------------------------------------------"
print "-----------Starting SIMULATION 3---------------"
print "------------------------------------------------"

foo=raw_input("Press Enter to start...")

for xSTA in range(30,1001,10):
    for ySTA in range(40,1001,10):
        A,n=0,0
        STA=[xSTA,ySTA]
        RSSI,propagation_time=propagation.propagation(STA,APs,f,Ptx,0.3)

        [A,n]=location.calibrate(RSSI,APs,STA,error)

        if A!=0 and n!=0:
            iterations=0
            correct=0
            #Sweeps the two axes
            for napx in range(20,1000,10):
                for napy in range(30,1000,10):
                    iterations=iterations+1
                    nSTA=[napx,napy]

                    #Simulates for each position of the sweep
                    nRSSI,propagation_time=propagation.propagation(nSTA,APs,f,Ptx,0.3)
                    distances=[]
```

```
            for i in range(0,len(nRSSI)):
                distances.append(pow(10,((   A-nRSSI[i]  ) / (  10*n  )    )))
            # print '\n STAx={0},STAy={1}'.format(napx,napy)
            x,y=trilaterate.tri4TOF(APs,distances, show = False)
            if (x != -99 and y != -99):
                op1= abs(x-napx)
                op2= abs(y-napy)
                if op1<150 and op2 <150:
                    correct=correct+1


    print ("-----RESULTS-----")
    print "Real STA position ({0},{1})".format(xSTA,ySTA)
    print "Iterations={0}".format(iterations)
    print "Correct positions={0}".format(correct)
    print "{0} %\n\n".format(100.0*correct/iterations)

    fi= open(fileName,"a")
    string='Real STA position (' + str(xSTA) + ',' + str(ySTA) + ')\n Iterations= ' + str(iterations) +
     '\n Correct positions= ' + str(correct) + '\n' + str(100.0*correct/iterations) + '%\n'
    fi.write(string)
    fi.close()

else:
    print "Error calibrating for STA position ({0},{1})\n\n".format(xSTA,ySTA)
```

Figura 63. Simulation 3: main.py

Simulation 4

```
'''
-------------------- main.py --------------------
It shows the main menu
Author: Jose Manuel Noguerol Diaz
-------------------- main.py --------------------
'''

import propagation
import locationHandler as location
import sys
import os,re
import trilateration as trilaterate
import datetime
import matplotlib.pyplot as plt
import numpy as np

# Clear the command line
os.system("clear")

Ptx=0
#error margin allowed
MARGIN=100

print "-------------------------------------------------"
print "-----------Starting SIMULATION 4---------------"
print "-------------------------------------------------"

fp=open('./coordinates.txt', 'r')
line = fp.readline()
cnt = 1
APsList=[]

while line:
    APsList.append(line.strip())
    line = fp.readline()
    cnt += 1
fp.close()

# Get current working directory
dir=os.getcwd()
pattern=".png"
for ff in os.listdir(dir):
    if re.search(pattern, ff):
        os.remove(os.path.join(dir, ff))
print "All previous PNG files have veen deleted"



print "\nLoaded {} iterations...".format(cnt)

foo=raw_input("Press Enter to start...")


for nAPs in range(4,10,1):

    f=[]
    f=5e9*np.ones((nAPs,), dtype=int)
```

```
#Arrays to append the results and the list od Lfading which it has been performed with
fadingList=[]
resultList=[]

for Lfading in range(0,11):
    iterations=0
    correct=0
    result=0
    #Sweeps the two axes
    for napx in range(15,1001,10):
        for napy in range(15,1001,10):

            iterations=iterations+1
            nSTA=[napx,napy]
            x=-99
            y=-99

            APs=location.parser(APsList[nAPs-4])

            # APs location won't be printed just to make the command line clearer.
            nRSSI,propagation_time=propagation.propagation(nSTA,APs,f,Ptx,0.3,Lfading)

            x,y=location.positionUsingWeights(APs,nRSSI)

            if (x != -99 and y != -99):
                op1= abs(x-napx)
                op2= abs(y-napy)
                if op1<MARGIN and op2 <MARGIN:
                   correct=correct+1

    result=round(100.0*correct/iterations,2)
    resultList.append(result)
    fadingList.append(Lfading)

    print "For {} APs, Lfading {}  ----> {} iterations and {} correct ones ({} %)\n".format(nAPs,Lfading,iterations,correct,result)

fig, ax = plt.subplots()
axes = plt.gca()
axes.set_xlim([0,10])
axes.set_ylim([0,108])
plt.plot(fadingList,resultList,linestyle='-', marker='o')

#Print the value of each point
for i,j in zip(fadingList,resultList):
    axes.annotate(str(j),xy=(i+0.01,j+3))
axes.set_title('Locating using weights -- Simulation 4 - Number of APs: '+str(nAPs))
axes.set_xlabel('Lfading')
axes.set_ylabel('% Of valid points (less than 1 m) ')
fig.savefig("Sim 4 "+"and "+str(nAPs)+" APs.png")
```

Figura 64. Simulation 4: main.py

```
'''
-------------------- propagation.py -------------------------
Let us get RSSI and propagation time based on an 802.11
pathloss model
Author: Jose Manuel Noguerol Diaz
-------------------- propagation.py -------------------------
'''

import numpy as np
# Esta funcion simula la propagacion de la senal WiFi en una habitacion
# grande. Los APs sondean a la STA y tras cierto tiempo (propagation_time)
# reciben una respuesta con cierta potencia (RSSI)

# % INPUTS ----------------------------------------------
# % STA: vector 2x1 con la posicion de STA.
# % APs: matrix de 2xN con la posicion de los N APs.
# % f: vector de 1xN con las frecuencias de cada AP en Hz.
# % Ptx: Potencia transmitida en dBm.
#
# % OUTPUTS ----------------------------------------------
# % RSSI: vector 1xN con las RSSI recibidas en los APs.
# % propagation_time: vector con los tiempos de ida y vuelta del mensaje que manda el AP.

def propagation (STA,APs,f,Ptx,coef,Lfading):
    scale=100
    c=3e8
    Lwalls=0
    N=len(APs)
    direction_vectors=[]
    distance=[]
    propagation_time=[]
    d=[]
    L=[]
    for i in range(0,N):
        direction_vectors.append(np.subtract(STA,APs[i]))
        distance.append(np.linalg.norm(direction_vectors[i])/scale)
        normal=np.random.random()
        propagation_time.append(2*(1+coef*normal)*distance[i]/c)
        d.append(max(distance[i],1))
        if (d[i]>5) > 0:
            L.append(40.05+20*(np.log10(f[i]/1e9)/2.4)+20*np.log10(min(d[i],5))+(d[i]>5)*35*np.log10(d[i]/5))
        else:
            L.append(40.05+20*(np.log10(f[i]/1e9)/2.4)+20*np.log10(min(d[i],5)))


    L_space=(np.add(L,Lwalls))
    RSSI=(np.subtract(np.subtract(Ptx,L_space),Lfading*np.random.normal(0,1,N)))
    return RSSI,propagation_time
```

Figura 65. Simulation 4: propagation.py

```
'''
---------------------- locationHandler.py ----------------------
Functions that let the user to use a location algorithm
Author: Jose Manuel Noguerol Diaz
---------------------- locationHandler.py ----------------------
'''

import math
import numpy
from sympy import Symbol
from sympy.solvers import solve
from sympy.functions import re
import time
import datetime
import trilateration
import os
import math

'''
-----positionUsingWeights----
Used to handle the fourth option (Weights)
Calculates them by using different formulas
'''

def positionUsingWeights(APs,RSSI):

    x=0
    y=0
    total=sum(RSSI)
    N=len(APs)
    a=[]
    b=[]
    weights=[]
    for item in RSSI:
        a.append(pow(10, 0-item/20.0))
    for item in a:
        b.append(sum(a)/item)
    for item in b:
        weights.append(item/sum(b))

    for i in range(0,N):
        x=x+(weights[i]*APs[i][0])
        y=y+(weights[i]*APs[i][1])

    return x,y

def parser (APsList):

    split=splitString(APsList)
    if len(APsList)>2:
        deleted=delete_(split)
    APs=buildString(deleted)
    return APs
```

```
def splitString(string):

    # Split the string based on space delimiter
    list_string = string.split(' ')

    return list_string

def delete_(string):
    deleted=[]

    for i in string:
        if i !='-':
            deleted.append(i)
    return deleted

def buildString(string):
    N=len(string)
    APs=[]
    for i in range(0,N,2):
        APs.append([int(string[i]),int(string[i+1])])
    return APs
```

Figura 66. Simulation 4: locationHandler.py